UNIVERSITY OF LJUBLJANA
SCHOOL OF ECONOMICS AND BUSINESS

MASTER'S THESIS

SUPPLY CHAIN OPTIMIZATION AND DIGITALIZATION AT UBIQUITY ROBOTICS

Ljubljana, June 2025 MATJAŽ BOŠTIC

AUTHORSHIP STATEMENT

The undersigned Matjaž Boštic, a student at the University of Ljubljana, School of Economics and Business, (hereafter: SEB LU), author of this written final work of studies with the title Supply chain optimization and digitalization at Ubiquity Robotics, prepared under supervision of Jure Erjavec, PhD.

DECLARE

1. this written final work of studies to be based on the results of my own research;

2. the printed form of this written final work of studies to be identical to its electronic form;

3. the text of this written final work of studies to be language-edited and technically in adherence with the SEB LU's Technical Guidelines for Written Works, which means that I cited and / or quoted works and opinions of other authors in this written final work of studies in accordance with the SEB LU's Technical Guidelines for Written Works;

4. to be aware of the fact that plagiarism (in written or graphical form) is a criminal offence and can be prosecuted in accordance with the Criminal Code of the Republic of Slovenia;

5. to be aware of the consequences a proven plagiarism charge based on the this written final work could have for my status at the SEB LU in accordance with the relevant SEB LU Rules;

6. to have obtained all the necessary permits to use the data and works of other authors which are (in written or graphical form) referred to in this written final work of studies and to have clearly marked them;

7. to have acted in accordance with ethical principles during the preparation of this written final work of studies and to have, where necessary, obtained permission of the Ethics Committee;

8. my consent to use the electronic form of this written final work of studies for the detection of content similarity with other written works, using similarity detection software that is connected with the SEB LU Study Information System;

9. to transfer to the University of Ljubljana free of charge, non-exclusively, geographically and time-wise unlimited the right of saving this written final work of studies in the electronic form, the right of its reproduction, as well as the right of making this written final work of studies available to the public on the World Wide Web via the Repository of the University of Ljubljana;

10. my consent to publication of my personal data that are included in this written final work of studies and in this declaration, when this written final work of studies is published.

11. that I have verified the authenticity of the information derived from the records using artificial intelligence tools.

Ljubljana, 19-6-2025 Author's signature: 

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF APPENDICES

LIST OF ABBREVIATIONS

SC	Supply chain
SCM	Supply chain management
API	Application programming interface
WMS	Warehouse management system
OFP	Order fulfilment process
SPOF	Single point of failure

1 INTRODUCTION

1.1 Background of the study

A supply chain is defined as a set of three or more entities (organizations or individuals) directly involved in the upstream and downstream flows of products, services, finances, and/or information from a source to a customer (Mentzer et al., 2001). Modern supply chain designs are dynamic, meaning they can adapt to changes in the company's business model. They are also resilient to common risks that they might face in the future (Beske, 2012).

Companies are becoming increasingly competitive. They are forced to act globally and constantly adapt to environmental changes. This forces them to manage and optimize their supply chains in new ways to reduce expenses and maximize revenues.

Supply Chain Management integrates key business processes from end-users through original suppliers that provide products, services, and information that add value for customers and other stakeholders (Lambert & Cooper, 2000).

A supply chain design is »the decisions regarding the number and location of production facilities, the amount of capacity at each facility, the assignment of each market region to one or more locations, and supplier selection«. Global supply chain design also includes selection of facilities at international locations (Meixell & Gargeya, 2005).

A supply chain consists of several interconnected components, including procurement, production, inventory management, and logistics. Within logistics—which plays a crucial role in ensuring goods move efficiently through the supply chain—transportation is a key activity. It involves the physical movement of goods from one location to another and often represents a significant portion of logistics costs, accounting for up to a third of total logistics expenses. (Tseng et al., 2005). There are many different means of transport. Those are transport by trucks, ships, trains, and planes. Each of them has its pros and cons. Another part of supply chains is warehousing, which represents the storage of goods. Warehousing optimisation is an important part of supply chain management because warehousing acts as a node in linking the material flow between the supplier and customer. WMS (warehouse management system) plays an important role in managing modern warehouses. It is a computer application that supports features like inventory level monitoring, managing processes, people and equipment, etc. (Ramaa et al., 2012).

A part of the supply chain – the order fulfilment process, and its automation, plays a key role in businesses using e-commerce. When someone places an order online, that request has to travel to the appropriate storage facility or warehouse. There, the product is located, packed, and dispatched for delivery to the shopper. This system must operate both swiftly and correctly. Delays or errors can lead to unsatisfied customers who may decide not to return.

An important issue that many businesses face is the automation of this process. Automated systems can process orders more rapidly, reduce human mistakes and labour costs, ensure products reach customers as expected, and reduce business operating expenses. Additionally, inventory tracking software is beneficial. It offers real-time updates on stock levels. So, when a shopper buys an item, the system can instantly confirm its availability in any of the company's warehouses. This avoids situations where someone buys a product that's out of stock. If many warehouses have the required item(s), this software also enables the possibility to make an automatic decision from which warehouse to send the item based on its corresponding expected shipping costs. Effective order handling and stock management are vital for online retailers. Implementing automation and using inventory tracking tools can substantially enhance both efficiency and customer satisfaction (Croxton, 2003; Shukla, 2021).

Proper design of the supply chains is especially important in fast-growing, high-tech industries since supplies, manufacturing, and transport in such industries are expensive and thus have to be managed properly. Robotics is one of those industries that is playing an increasingly important role in our lives. It is a multidisciplinary field which requires the integration of software and hardware. Robot arms, or manipulators, for example, are very useful in the production industry. However, this branch of robotics has a fundamental disadvantage, which is its lack of mobility (Siegwart et al., 2011). Another important branch of robotics is mobile robotics. Mobile robots can move around in their environment. This makes them appropriate for various applications, such as automating repetitive tasks that could be otherwise performed by humans, like moving things around, or for applications in environments that would be otherwise inaccessible or hostile to humans.

Robotics plays an increasingly vital role also in supply chain management by improving efficiency, accuracy, and speed in key processes such as warehousing, inventory handling, and order fulfillment. Automated systems reduce human error, lower labor costs, and enable faster response times, making supply chains more agile and resilient in the face of growing demand and complexity.

In my thesis, I will consider the company Ubiquity Robotics, which is a US-based manufacturer of general-purpose mobile robots. The company was founded in 2017, and because of rapid growth, it is facing a problem with organising the supply chain for its products. Currently, the company has only one warehouse from which orders are sent all across the world. This results in high shipping costs for countries that are far away from the warehouse. This is especially inconvenient for the company since its products are relatively big and heavy, meaning that shipping costs are high even if the buyer is not that far from the warehouse. Another problem that the company is facing is its order fulfilment process, which requires quite a significant amount of human labour since it is not automated enough. This results in high operating expenses.

As aforementioned, the company considered produces mobile robots. The base robot called »Magni« can have various user-specific applications. It is a platform on which users can build almost whatever they desire without the need to do the part of the robot that handles mobility. Applications include a warehouse robot, a telepresence robot, a disinfection robot, a cocktail waiter robot, a person-following robot, and others.

1.2 Definition of the problem

In the field of robotics manufacturing and manufacturing in general, a proper supply chain is crucial. After analysing the supply chain of the company considered, which produces relatively large and heavy products, two main challenges have been identified.

The first issue is about distribution. Using just one warehouse for all their storage and worldwide shipping makes shipping costs high. This is especially true for deliveries going to places far from this main warehouse. The large size and weight of the robots add even more to these shipping costs.

The second problem arises when considering how orders are processed. The company mainly relies on the administrative assistant to manually handle orders through web applications, using only a small amount of automation. This means there is a higher chance of mistakes, and it costs more because constant manual labour is needed.

To sum it up, our analysis of the supply chain showed two big issues: the costs of using only one warehouse and the inefficiencies of mostly manual order handling. This thesis outlines the analysis used to identify key problems, then examines them in depth to develop and implement effective solutions.

1.3 Aims and goals

This thesis aims to showcase an approach to identifying improvements in the company’s supply chain and providing solutions.

The main goal of the thesis is to propose valuable improvements in the supply chain for the aforementioned company, which will reduce SC costs while maintaining the level of customer service. Some of the solutions will also be implemented and described.

All research goals are:

- Research and present the theory behind supply chains and the process of finding improvements in them. Also, the theory around low-code language – Deluge and how to automate the order fulfilment process with it.
- Deeply analysing the selected company's supply chain

- Explaining the current state (AS IS) and future state (TO BE) of the company's Chinese supply chain and determining problems in the current state, especially the ones relating to transport, warehousing, and information flow.
- Finding solutions to problems in the current supply chain state with the help of the A3 problem-solving technique.

1.4 Scope and limitations of the study

The study's scope is the considered company's current supply chain (with a focus on the order fulfilment process) and its optimization.

1.5 Significance of the study

This study will be important to the company being considered. As a result, we expect a significant reduction in the current supply chain cost.

The study will also be helpful for companies in a similar situation to ours – those that need to optimize their supply chains and/or automate the order fulfilment process using a low-code platform.

1.6 Research methodology

The methodology of the supply chain analysis used in the thesis is value stream mapping (VSM) because it is the most widespread technique for tackling the problems I am facing. I will perform VSM with the help of the general manager and administrative assistant. To move from the current SC state to the proposal of the future SC state, I will help myself with the A3 problem-solving technique.

Value stream mapping (VSM) is a useful technique for mapping the supply chain to reduce its cost and time afterwards. For this project, I will apply this process to a company I work in: Ubiquity Robotics. First, I will draw the VSM map of the current supply chain state, identify problems in this state, and then draw the map of the potential future state.

Another tool used in the thesis is the A3 technique, named after an A3-sized paper and often used to solve problems. The steps of this technique are the following: Issue, Background, Current condition, Goal, Root cause analysis, Target condition, Countermeasures, Implementation, and cost analysis, Test, and Follow-up/audit (Bassuk & Washington, 2013).

For modelling VSMs and performing the A3 technique, I will conduct interviews and collaborate with the general manager and other employees (especially the administrative assistant) to arrive at the desired goals. There is currently no proper internal documentation and models of the supply chain or business processes, so I will (with the help of co-workers) have to make them from scratch.

1.7 Organization of the thesis

The rest of the thesis is organised as follows. First, a literature review will be performed, where the most important concepts and tools will be described based on already existing literature. After that, the research methodology will be presented, which will also include a description of how the data with which we deal in this thesis was collected. Next, the results will be described. In this section, the current and future supply chain state will be presented. These SC states will be presented with the VSM approach. The following part will be the presentation and analysis of the developed Deluge code, which implements the automation of the order fulfilment process. Finally, the discussion and conclusion will follow. The interpretation of the main findings and contributions will be described here, along with a conclusion and recommendation for further research.

2 LITERATURE REVIEW

2.1 Supply chain management

Nowadays, the competition between companies doesn't happen in terms of businesses as individual and independent entities but rather in terms of their supply chains. Thus, in such an environment, the success of a business depends not only on its performance but also on its ability to integrate its network of business relationships (Lambert & Cooper, 2000).

»Supply chain management (SCM) can be defined as the monitoring and optimization of the production and distribution of a company’s products and services. It seeks to improve and make all processes involved in turning raw materials and components into final products more efficient and getting them to the ultimate customer. Effective SCM can help streamline a company's activities to eliminate waste, maximize customer value, and gain a competitive advantage in the marketplace« (Fernando, 2022).

In the introduction, I also cited the definition of the term »supply chain« as a set of three or more entities (organizations or individuals) directly involved in the upstream and downstream flows of products, services, finances, and/or information from a source to a customer (Mentzer et al., 2001). As evident from this definition, some organizations divide the supply chain into two main parts, upstream and downstream, to simplify their management. The upstream part comprises all activities related to the organization's suppliers. On the other hand, the downstream part refers to post-manufacturing activities, in other words, the ones that distribute the product to the final customers (Britt, 2021).

Another way of dividing the supply chain into two parts is into material and information flow. Material flow represents the flow of physical supplies and products from one part of the SC to another, and information flow represents the flow of information between different parts of the supply chain.

While the term »supply chain management« doesn't have a single definition, its definitions by various authors can be categorized into three categories: a management philosophy, implementation of a management philosophy, and a set of management processes. As a philosophy, SCM views the supply chain as a whole (a single entity), where the total flow of goods from the supplier to the ultimate customer is managed. It attempts to converge intrafirm (within the firm) and interfirm (between multiple firms) operational and strategic capabilities into a unified whole. The philosophy also has a customer-focused characteristic, which is to create unique sources of customer value, leading to customer satisfaction. To adopt such a philosophy, the company must establish some management practices (activities). Those activities are integrated behaviour, mutually sharing information, mutually sharing risks and rewards, cooperation, the same goal and the same focus on serving customers, integration of processes, and partners to build and maintain long-term relationships. Usually, not all of these activities are performed by a single firm to maximize customer value; thus, forming strategic alliances with other firms provides a competitive advantage through creating customer value. Instead of focusing on activities, other authors have focused on management processes. A process is a specific ordering of work activities across time and place, with a beginning, an end, clearly identified inputs and outputs, and a structure for the act. It was proposed that to successfully implement SCM, all the functions within a supply chain need to be reorganized as key processes which focus on meeting the customer's requirements. The firm needs to be organized around these processes (Ballou, 2007; Mentzer et al., 2001).

Skjoett-Larsen (1999) was one of the first to create an article about supply chain management. In 1999, the term “supply chain management” started to gain traction among managers and scholars. This work was done because the concept was not yet well defined at the time. Among others, the work also introduces three different theoretical perspectives on supply chain management: the transaction cost approach, the network perspective, and the resource-based view.

Once the supply chain is operational, a business department called “Control tower” usually emerges, overseeing and managing day-to-day supply chain operations. It keeps an eye on things like inventory levels in both the company’s warehouse and the suppliers. The more automated the supply chain is, the less work from the side of the Control tower is required. The control tower can run into various problems if the supply chain is not developed well enough, one of them being the lack of visibility. This means that there isn’t sufficient / fast enough oversight over some parts of the SC. For example, it might take a few days to get information about the stock level from a certain supplier after the enquiry, which slows down the entire supply chain operation. This delay becomes a so-called “bottleneck” in the SC.

2.2 Supply chain design

Supply chain design considers problems like the number and location of production facilities, the amount of capacity at each facility, the assignment of each market region to one or more locations, and supplier selection for sub-assemblies, components and materials. If we talk about the design of *global* supply chains (the problem with which we deal in this thesis), that will also include facilities at international locations and globalization problems that come with it (Meixell & Gargeya, 2005).

Supply chain design consists of those decisions that influence the investment patterns made by the company across its various supply chains. These decisions determine the kinds of problems that the supply chain can address and those that it can't address (yet). They also affect the relationships that emerge between supply chain partners, the vulnerability, and the degree of visibility of the supply chain. Besides, they also determine how well the supply chain abilities match the strategic objectives of the firm (Melnyk et al., 2014).

Globalization of supply chains has its advantages, but it also has some drawbacks. The pros are the reduction of costs, increased revenues, and also improved reliability (reduced risks in case one or more elements of the supply chain fail because there are other elements in other parts of the world with similar functionality to the failed element that can replace it). The most obvious way of achieving increased revenue stems from the fact that globalization offers access to larger (global) markets. A lower cost of labour can be achieved if we set up a foreign factory in a location that has a lower cost of workforce. If we carefully pick warehousing locations, taxes and shipping costs can be reduced significantly. Shipping costs can be reduced because we can ship in larger quantities to a foreign warehouse, for example with ship freight and after that with trucks to individual customers. This method is much cheaper than, for example, global air freight from a single location in the world. But, as aforementioned, there are also cons of global supply chains in comparison to domestic ones. One problem is that large geographical distances increase transportation costs and lead times, which lead to greater inventory costs. Another is that if we move supply chain elements to developing countries where the workforce is cheaper, we might face lower work quality and lower supplier availability and quality. There might also be infrastructural deficiencies in transportation and telecommunication. Global supply chains also carry risks of variability and uncertainty in currency exchange rates (which affect the price paid for goods that are purchased in the supplier’s currency), economic and political instability, and changes in the regulatory environment (Meixell & Gargeya, 2005).

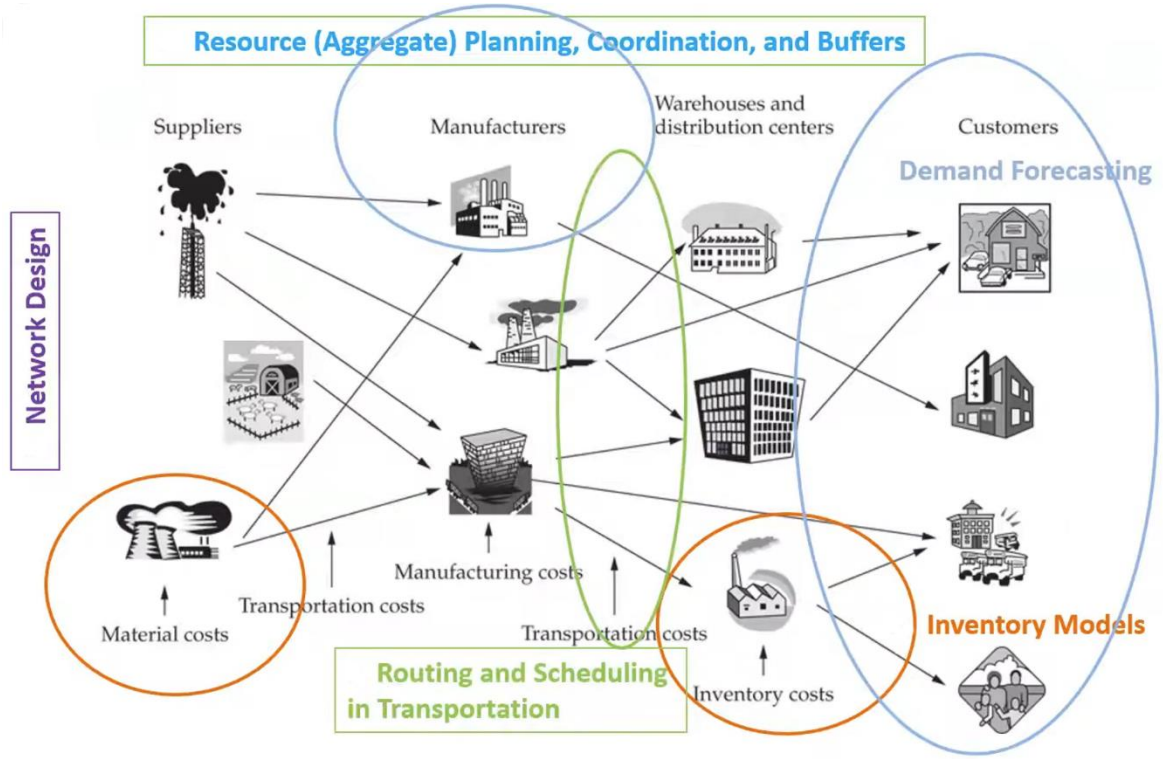
Schrauf & Berttram (2016) examine the evolving design of supply chains in the context of Industry 4.0, characterized by the increasing digitalization of enterprise processes. They argue that conventional supply chains are transitioning into integrated, intelligent, and highly efficient ecosystems. Traditional supply chains typically consist of distinct functional areas, including marketing, product development, and manufacturing, each operating with varying degrees of coordination. Digitalization blurs the boundaries between these entities,

converting them into an integrated ecosystem that is fully transparent to everyone involved - from the suppliers of raw materials and components to the transporters of those supplies and finished products and, finally, to the end customers. The result of newly developed/developing technologies will enable companies to react to disruptions in the supply chain and even anticipate them by fully modelling the network, creating "what-if" scenarios, and adjusting the supply chain in real-time as conditions change. Thus, once built, Industry 4.0 will offer a new degree of resiliency and responsiveness, potentially enabling its early adopters to beat the competition in the effort to provide customers with the most efficient and transparent service (Schrauf & Berttram, 2016).

Lancioni et al. (2000) did a similar thing of discussing how an industrial revolution impacted supply chain management and design. In their case, this was the Industry 3.0 revolution, which happened more than 20 years ago, when computers and the internet were getting increasingly more prominent.

The design of supply chains needs to adapt to various factors, especially to the product type. For example, large and heavy products need supply chains designed differently than small and light products. Payne & Peters (2004) made a research on how significant is the result of designing the supply chain tailored to the product type. It revealed significant value in matching specific product clusters with appropriate supply chain designs and that any mismatch represents supply chain underperformance.

Figure 1: Elements of supply chain design



Source: Pazour (2021)

In Figure 1, elements of the supply chain are presented.

2.3 Understanding the order fulfilment process

Order fulfilment is a fundamental component of retail operations, particularly in the e-commerce sector. It represents the full process from receiving an order from the customer (e.g., on a web store) to delivering the product to the buyer. The efficiency and effectiveness of this process significantly impact customer satisfaction and operational costs.

»The order fulfilment process describes the execution of various activities to complete an order. In the context of production, this usually includes the acquisition of (customer) orders, purchasing secondary requirements, production, shipping and post-delivery activities. The process varies depending on the type of company and the order fulfilment strategy« (Hiller et al., 2024).

There is a series of steps in this process, the key ones being:

1 Order Receipt: This initial step acknowledges the receipt of an order. Once a customer places an order, the seller's system logs it, typically through an automated platform that integrates with the online shop. This system ensures the order details are accurate and prepares the seller to move to the next stage. Suppose there are multiple warehouses (fulfilment centres) from which the order can be shipped. In that case, the most appropriate one is selected based on the shipping information and the amount/types of items to deliver.
2 Picking and Packing: Once the order is received and successfully processed, the next step is to locate the product within the storage facility. In a warehouse setting, this task involves "picking" the correct item from its storage location. This process can be streamlined using barcode/QR code systems or RFID technology to ensure accuracy. After picking, the product is "packed" securely to prevent damage during transit.
3 Shipping: After packing, the product is handed over to a shipping carrier. This step involves labelling the package with the appropriate shipping information and selecting a shipping method that balances cost and speed. Efficient shipping processes are crucial for meeting delivery deadlines and customer expectations.
4 Delivery and Returns: The final step is delivering the product to the customer. However, the process does not end here. Sellers must also have a clear returns policy and efficient return handling process for products that customers wish to return due to defects, dissatisfaction, or other reasons. A streamlined return process enhances customer trust and satisfaction.

In Figure 2, a bit more detailed/general steps of the OFP are presented.

Figure 2: A bit more detailed/general steps of the OFP

Source: Shukla (2021)

There is room for optimization in each of these steps. For example, e-commerce retailers often face the challenge of assembling large numbers of time-critical orders, each consisting of just a few items ordered (low order quantities). There are many ways in which the duration of order picking can be reduced, for example, with automated picking workstations, robots, etc. (Boysen et al., 2019).

Now, the pros and cons of automating the order fulfilment process (OFP) will be discussed.

Pros:

- Increased efficiency: Automation speeds up the order fulfilment process by reducing manual tasks, thereby increasing the overall efficiency and decreasing the overall order fulfilment time.
- Cost reduction: While the initial setup of automation systems can be costly, over time, it reduces labour costs and errors, leading to significant savings.
- Scalability: Automated systems can easily handle large volumes of orders, making it easier for businesses to scale operations without a proportional increase in labour costs.

- Improved reliability: Automation minimizes human error in order processing, ensuring that customers receive the correct items.

Cons:

- High initial investment: The setup cost for automation technology, including software, machinery, and infrastructure, can be unreasonably high, especially for small businesses.
- Complex implementation: Integrating automated systems into existing processes requires time and expertise, which can disrupt operations during the transition period.
- Maintenance and updates: Automated systems require regular maintenance and updates, which can incur additional costs and require specialized knowledge.

One significant benefit of optimizing the order and inventory management process is the potential for increased standardization, which helps reduce operational variability. The order fulfilment IT system oversees the orders and standardizes and integrates the process across the planning, purchasing, and production stages. Specifically, it encompasses order management (including order entry, automation, and payment processing), resource planning (such as SKU management, facility design, and warehouse operations), production scheduling (covering master, weekly, and daily schedules), capacity and material planning (including inventory planning, replenishment, and product flow), production control (through supply chain management), packaging and shipping (covering shipment consolidation, tracking, notifications, and routing), as well as other functions like profiling, sharing best practices, strategic information, and supplier monitoring. In short, it »forces« the company to evaluate and standardize various components of its supply chain (Heydari et al., 2020).

»Even though the operational order fulfilment process is usually considered separately from the other supply chain processes and is believed to be conducted within the logistics function, it is not separable from the other business processes. Therefore, to understand order fulfilment, it is necessary to focus on the interaction with the whole supply chain system« (Heydari et al., 2020).

In conclusion, paying attention to the order fulfilment process is essential for ensuring customer satisfaction and operational efficiency in retail, especially in e-commerce. Its automation offers significant benefits in speed, cost, and accuracy, but it also involves considerable initial investment and complexity. Businesses must weigh the pros and cons carefully, considering their specific needs, order volume, and available resources, to decide on the best approach to order fulfilment and the amount of automation involved.

2.4 Value stream mapping

Value stream mapping (VSM) is a useful technique to reduce a company's supply chain cost and time.

Value stream maps are divided into two parts. One is the information flow, and the other is the material flow. That's why VSMs are often called material and information flow analysis (MIFA). Information flow represents how customers, suppliers, warehouses, the company itself and other supply chain entities communicate with each other. On the other hand, material flow represents the flow of goods between those entities.

The VSM originates from Toyota, where they first drew those diagrams.

VSMs are usually drawn so that suppliers are on the top left side of the map and customers are on the top right side. This gives a good overview of the material and information flow between them and other SC entities.

The goal of drawing current state maps and future state maps is to create afterwards an implementation plan of the changes that will need to be done to arrive from the current state to the future state and, after that, of course, execute that plan.

The activities in the supply chain are divided into value-adding and non-value-adding. Value-adding activities are those that directly benefit the customer, like assembly, welding, and painting the product. Non-value-adding activities are those that do not directly benefit the customer, such as meetings, movement of goods, dealing with errors in the processes, etc. (Tsonev, n.d.).

The implementation plan usually consists of changes like removing waste from the supply chain, which are activities that don't add value to the customer and aren't necessarily needed in the SC. It also includes automating activities that are currently done manually or in a non-optimal way.

Material flow can be either of push or pull type. Push means that the products are manufactured and shipped by predicted customer demand. Based on this predicted demand, the products are manufactured in larger batches and then shipped further down the supply chain. This leads to big inventories, which is not ideal and does not obey the "lean" way of manufacturing. Often, a better and more »lean« way of dealing with material flow is the "pull" way. This means that the products get produced and shipped based on the need for these products further in the supply chain. The pull begins with actual customer demand. In this way, there is less inventory, meaning less risk of producing products without customer demand.

The VSM is a lean tool for managing a company that increases value for the customer while eliminating waste and requiring fewer resources. Lean is a practice that consists of

continuous experimentation to achieve the highest value with zero waste. The first step in lean is determining what is valuable for a customer, in other words, figuring out what problem the customer needs to solve (Lean Enterprise Institute, n.d.).

Lean manufacturing is also known as the Toyota Production System (TPS). This production strategy aims to eliminate waste and deliver increased value to the product and customer. Generally, lean manufacturing focuses on discovering the major sources of waste and eliminating that waste using lean manufacturing tools. Other lean manufacturing tools have also been extensively used to identify and eliminate waste (besides the VSM). These are Kanban, 5 Sigma, Kaizen, Total Productive Maintenance, PokaYoke, etc. Lean manufacturing attracts great attention in manufacturing industries due to its numerous advantages (Ahmad et al., 2017).

The findings of a study (Ahmad et al., 2017) which (among other things) dealt with evaluating the results of implementing VSM in small and medium enterprises, indicate that value stream mapping is a practical approach to eliminating waste and improving overall productivity.

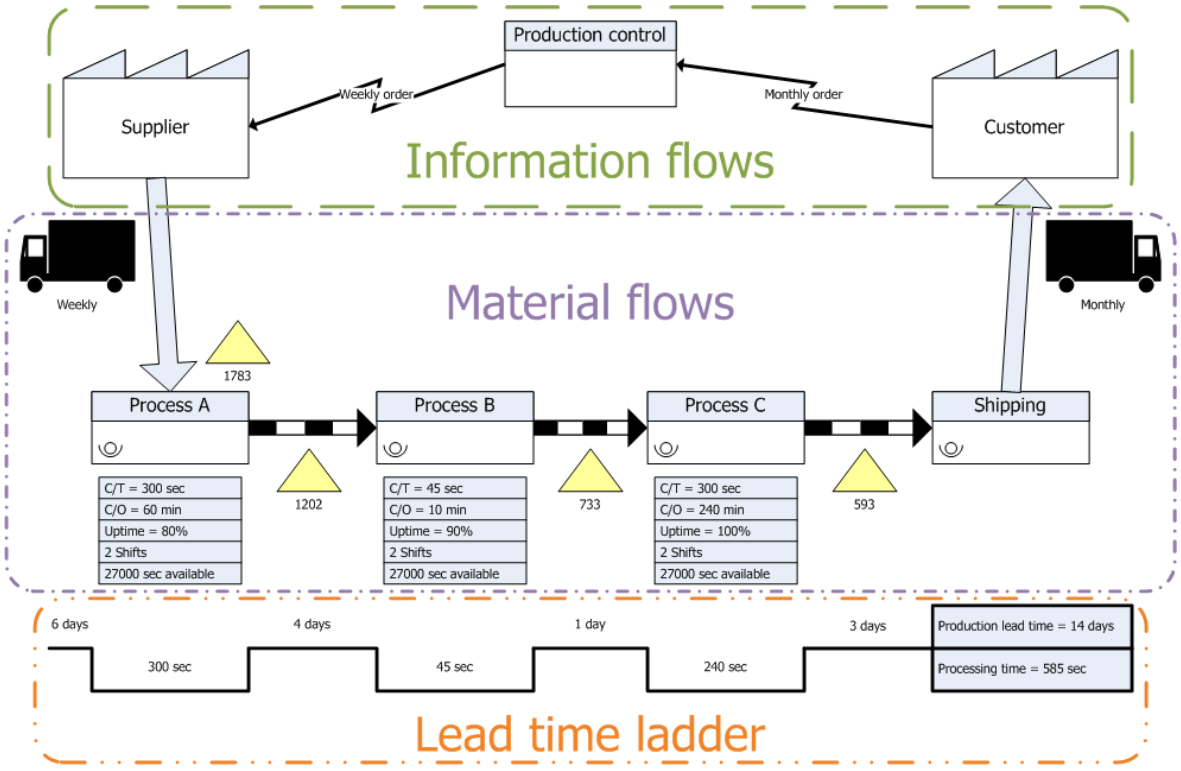
»As with any lean management toolset, the principle aim of Value Stream Mapping is to improve processes. This is achieved by highlighting areas of waste generation within a process, enabling businesses to eliminate these activities. Value Stream Mapping also has the benefit of categorizing process activity into two main areas: value-add, non-value-add (but necessary)/waste…« (Duranik et al., 2011).

Singh et al. (2010) discussed the lean implementation process and its resulting benefits with the help of VSM. They presented a case of a company in the production industry. The results are presented in the following paragraph.

»After comparison of the current and future state of shop floor of the selected industry it is found that reduction in lead time was 83.14 percent, reduction in processing time was 12.62 percent, reduction in work-in-process inventory was 89.47 percent, and reduction in manpower requirement was 30 percent. The rise in productivity per operator was 42.86 percent« (Singh et al., 2010).

In Figure 3, an example VSM is presented.

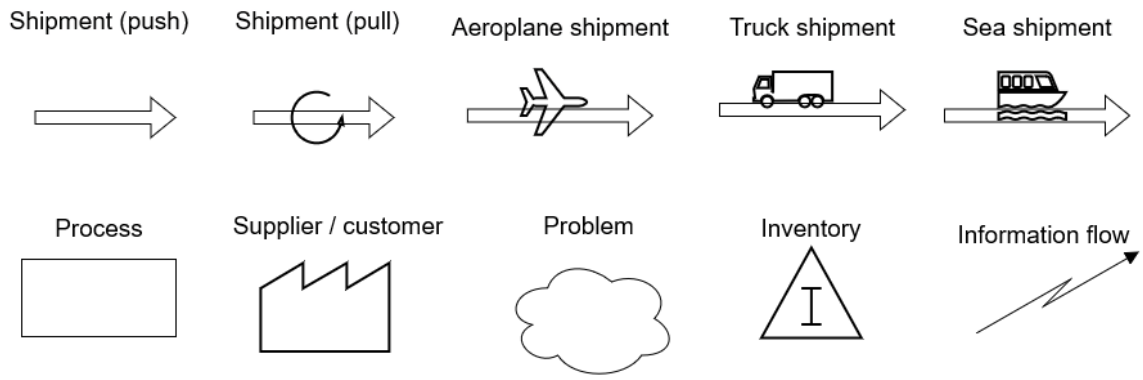
Figure 3: Representation of a Value Stream Map



Source: Penfield (2013)

On Figure 4, VSM symbols used later in the current and future VSM maps are explained.

Figure 4: VSM symbols



Source: Own work

2.5 A3 problem-solving technique

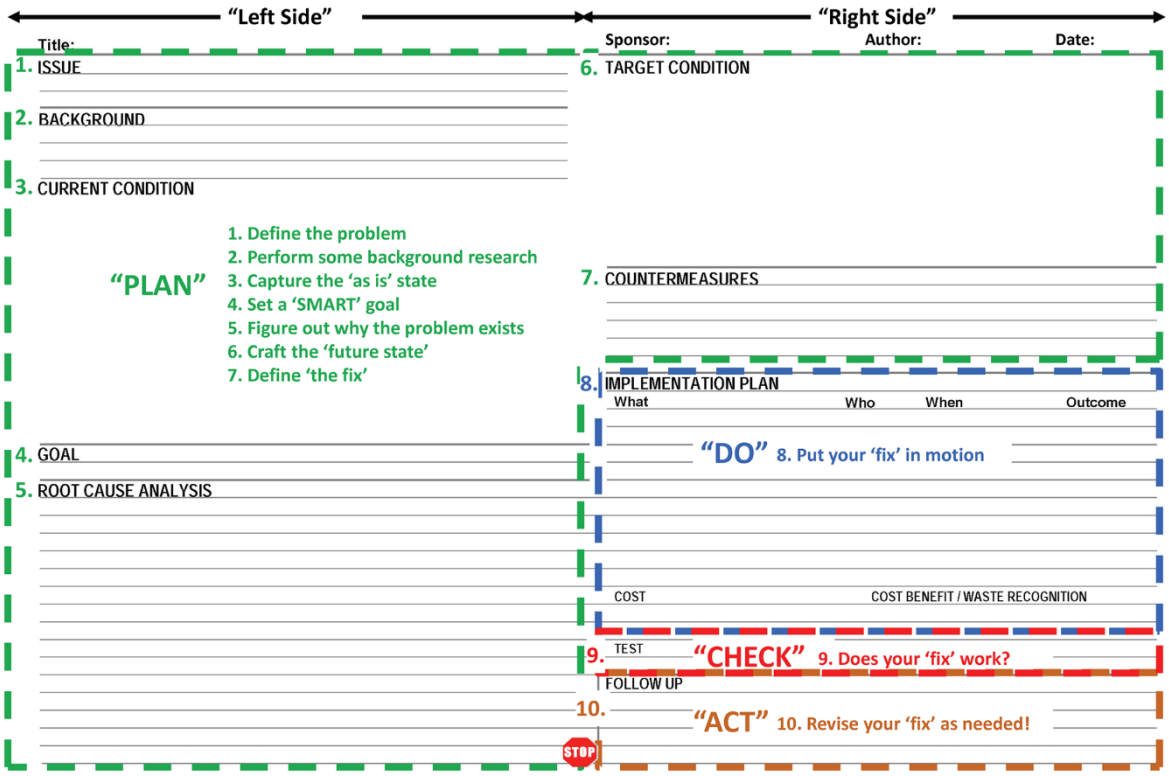
The Plan-Do-Check-Act (also known as the “quality loop”) was a predecessor to the A3 technique and was developed in 1920. Walter A. Shewhart introduced the "Plan-Do-See" (PDS) concept, which was later expanded upon by W. Edward Deming, an important figure in quality management at the time. Deming developed the "Plan-Do-Check-Act" (PDCA) cycle, also known as the "Deming cycle." This methodology was introduced to Japan in the early 1950s and later to China in the late 1970s. Today, it is extensively applied, especially in the healthcare sector, to enhance medical quality and standardize practices, proving its effectiveness in modern hospital quality management.

The A3 technique emerged later, and now it’s one of many techniques for problem-solving. It was first used and developed by Toyota company. Just like VSM, A3 is also a lean technique. Its primary purpose is to find consensus on complex decisions. It was named after A3 paper size because that paper size is usually used to draw an A3 report. This technique is based on the Plan-Do-Check-Act cycle, which is used by many problem-solving techniques (Bassuk & Washington, 2013). On Figure 5, the template for performing the A3 technique is displayed.

»A3 Thinking is fundamental to Toyota's benchmark management philosophy and their lean production system. It is used to solve problems, gain agreement, mentor team members, and lead organizational improvements. A structured problem-solving approach, A3 Thinking builds improvement opportunities through experience« (Anderson et al., 2011).

A3 Thinking and Reports, initially developed for problem-solving and process improvement in manufacturing settings, have shown their effectiveness in various fields, including healthcare. With its structured approach to identifying and addressing issues, this methodology has been successfully adapted far beyond its original context. For instance, in higher education, A3 reports have been employed as a teaching tool in healthcare executive MBA programs to convey essential skills in process improvement. This adaptation highlights the versatility of A3 Thinking, displaying its utility in creating a culture of continuous improvement and systematic problem-solving across various sectors (Anderson et al., 2011).

Figure 5: A3 template



Source: Bassuk (2013)

As seen from Figure 5, the technique usually has 10 steps. In the remainder of this section, these steps will be described (Bassuk & Washington, 2013; Lynn, n.d.).

Issue

A high-level statement that defines the problem. It has to be clear and focused. It intends to summarize the problem with which we will be dealing.

Background

A detailed description of the problem focused on parts that can't be described in the Current Condition step.

Current Condition

The current situation is represented as a drawing. It should fully describe the current situation. From this drawing, we realize what improvements may be necessary and should follow. First, careful observation of the work process is needed, resulting in a document with these observations. Then, the group responsible for the A3 should gather around the whiteboard/A3 paper and discuss each step in the process. If possible, assess the size of the problem and graph the data.

Goal

Defines how the improvement(s) will be measured and sets goal(s) based on that definition. The goals should be SMART (specific, measurable, attainable, relevant and timely).

Root Cause Analysis

In this step, we try to identify the root cause of the problem. We can do that by repeatedly asking »why« the problem occurs until we arrive at the root cause(s). We usually ask ourselves »why« five times. Ask yourself questions like:

- Where do we have a problem occurring because of communication breakdowns (issues with information flow)?
- Where are visible long delays without activity?
- What additional information do we need to make the problem more transparent?

Another way of determining the root cause(s) is with the fishbone diagram.

Target Condition

The target condition conveys what is necessary to meet the goal. It is a drawing of what the situation will look like once the necessary improvements are implemented.

Countermeasures

Description of improvements that need to be implemented to arrive to the Target Condition. Countermeasures should aim to:

- Define the intended outcome and the plan for achieving it.
- Arrive to clear, direct connections between people responsible for steps in the process.
- Reduce/eliminate loops, workarounds, and delays.

Implementation and Cost Analysis

Breakdown of improvements into specific tasks that will lead to these improvements. In this step, ownership, timelines, and expected outcomes for these tasks are described. We can tell from the expected outcomes whether the improvements were successful or not.

This can also include cost analysis, which determines the economic feasibility of improvements based on the reduction (increase) of costs resulting from them.

Test

This step determines whether measured results match the predicted results based on a pilot study carried out in 1-2 weeks.

Follow Up/Audit

This step describes the audit plan. This is a plan for the inspection of the financial information of a company, which leads to a better understanding of the success of the conducted problem-solving. This step also describes standard work based on how to implement the improvements.

2.6 Basics of programming in Deluge

2.6.1 Low-code software development

Traditionally, computer programming meant using textual programming languages such as C, Java, or Python by professionals with extensive experience in the industry. Low-code programming presents an alternative: letting citizen developers create programs using visual abstractions, demonstrations, or natural language, as well as code that closely resembles it. Hirzel (2022) recognized a gap between rising attention to low-code programming in the industry and the lack of literature on the topic. That's why he made an article that brings together low-code literature from various research fields, explaining how techniques work while providing a unified point of view

Businesses (besides individuals) may have various reasons for adopting low-code platforms. Low-code platforms can alleviate the shortage, costs, and time consumption of highly experienced developers and reduce mistakes in monotonous manual tasks. Another factor driving low code is the rise of cloud-based software as a service, which provides both more interfaces to automate and a platform to deploy the code.

Since Ubiquity Robotics uses Zoho as their business-management software solution, it makes the most sense to use Zoho’s low-code scripting capacity to automate the order fulfilment process.

Zoho can be considered a part of the so-called “rapid business application development« (RBAD) paradigm, which enables non-technical developers to create applications using interactive and feature-rich tools via browser without caring about the underlying infrastructure. Cui et al. (2011) wanted to better understand RBAD platforms' feasibility compared to traditional software development, so they conducted a user study with 12 participants creating applications on three selected RBAD platforms. Analyzing their behaviors and feedback from different development perspectives, they designed a conceptual business application model for RBAD platform, identified the best practices and gaps, and proposed 6 design recommendations for RBAD systems (Cui et al., 2011).

2.6.2 Introduction to Deluge

Zoho Corporation is an established name in business software. Founded in 1996, it is a global enterprise known for its diverse range of business tools. Their offerings span CRM systems, office utilities, IT oversight, and team collaboration platforms. Operating worldwide, Zoho aims to deliver cost-effective, cloud-driven solutions tailored to meet the demands of businesses, irrespective of their scale.

One of the very powerful things Zoho provides to its users is the Deluge or Data Enriched Language for the Universal Grid Environment. This scripting language was designed to enable better integration between Zoho's suite of applications and allow for a certain level of customization based on specific business needs. It is also often used to integrate Zoho's applications with other network applications, like web stores, warehouse management information systems, etc. Integrating different Zoho applications or connecting them to external network applications is mainly done with APIs (Application Programming Interfaces). I will be using this functionality a lot to achieve some of the goals of the thesis. That's why I will touch on the basic concepts of APIs and how they can be used with Deluge in this theoretical chapter.

Deluge's functions can be broadly categorized into:

- Integration**:** Many businesses utilize multiple tools to handle various aspects of their operations. For these tools to be effective, they should be capable of interacting and sharing data smoothly. Deluge facilitates this interaction within the Zoho ecosystem. It provides scripting capabilities, allowing data from one Zoho application to be accessed and used by another and connect to external network applications. This integration reduces the potential for errors from manual data transfer and helps maintain consistency in data across different applications. It also reduces the need for manual human labour, resulting in reduced costs of business operations.
- Customization**:** Businesses differ in their operational methodologies and requirements. A software tool that might be optimal for one business might need slight adjustments for another. Deluge provides a platform for businesses to make these adjustments to Zoho applications, allowing them to better align the software with their specific use case.

2.6.3 Advantages of low-code platforms

Automating parts of the order fulfilment process with low-code solutions like Zoho Deluge can significantly enhance efficiency, reliability and accuracy ("correctness" of the data or actions). Low-code platforms provide a user-friendly interface that allows businesses to automate complex tasks with minimal coding knowledge, making advanced technology accessible to small and medium-sized enterprises.

One critical area where automation can be applied is inventory management. Traditionally, inventory management requires constant monitoring and manual updates to ensure stock levels are accurate and products are available for order fulfilment. With Zoho Deluge, businesses can automate these tasks by setting up workflows that update inventory levels in real-time as orders are processed. This reduces the risk of human error and ensures that the inventory data is always up-to-date. For example, when an order is placed, the system can automatically deduct the ordered quantity from the inventory, alerting managers if stock levels fall below a certain threshold. This proactive approach helps maintain optimal inventory levels and prevents stockouts or overstocking.

Another vital aspect is forwarding orders to the appropriate warehouse or fulfilment centre. In businesses with multiple warehouses, determining the best location to ship an order from can be a complex decision involving shipping costs, delivery times, and stock availability. Zoho Deluge can streamline this process by creating automated rules that analyse these variables and assign orders to the most suitable warehouse. For instance, if a customer orders, the system can automatically check which warehouse is closest to the delivery address and has the required items in stock. It can then forward the order to that warehouse, optimizing the shipping process and reducing delivery times and costs.

By implementing such low-code automation solutions, businesses can achieve a more responsive and efficient order fulfilment process. Automating inventory management ensures that stock levels are accurate, improving customer satisfaction. Meanwhile, the automated forwarding of orders to the appropriate fulfilment centres helps reduce delivery times and shipping costs. These improvements enhance operational efficiency and provide a competitive edge in the market by enabling quicker and more reliable service.

Leveraging low-code platforms like Zoho Deluge to automate order fulfilment processes can transform how businesses handle inventory and order processing. It simplifies complex tasks, reduces errors, and allows for a more scalable and efficient operation, ultimately leading to increased customer satisfaction and business growth.

As the world of business software continues to grow and evolve, the demand for tools that can adapt to various needs has become apparent. The development of Deluge signifies a trend in the domain of business software. It points out the importance of providing its users with tools with various features and ensuring they can be integrated easily and adjusted to fit individual requirements. While the one-size-fits-all approach has advantages, the modern business environment often demands a more adaptable solution. Deluge was Zoho's response to this demand, aiming to ensure that their software tools can work more cohesively with one another (and with external applications) and be adapted to particular use cases.

In conclusion, Deluge's introduction to Zoho's offerings addresses specific needs within the business software landscape. It focuses on ensuring that different tools within the Zoho suite can communicate effectively and that these tools can be modified to better fit individual

business requirements. While it is a proprietary solution developed by Zoho, its existence also shows the growing need for adaptability and integration in business software tools today.

2.6.4 Syntax and Basic Constructs

In the field of programming languages, understanding syntax and the basic constructs is the foundational step to being able to use any language. Deluge, as a scripting language embedded within the Zoho ecosystem, possesses its unique structure and principles. However, its basic syntax is not very different from that of some mainstream programming languages, like C or Java. This section will be easy to understand for those familiar with other programming languages, but for readers without any pre-existing programming knowledge, it could be difficult. If it is too difficult for you, I would suggest referring to some basic programming tutorial(s) you can find online.

Basic Syntax

Every language has its grammar, a set of rules that define how statements should be formulated. Deluge's syntax resembles many mainstream programming languages but still has some unique traits.

Statements: Most instructions in Deluge end with a semicolon (;). This denotes the conclusion of a command, signalling the system to execute the said instruction.

Variables: A variable is a name used to store and track information in a program. Assigning values to variables in Deluge is straightforward. For instance, if one wishes to assign the number 10 to a variable named 'x', it would be x = 10.

Comments: Comments that annotate the code without being executed are initiated with double slashes (//). Any text following these slashes on the same line will not be executed as a part of the script.

Function Invocation: In Deluge, invoking functions follows two distinct patterns:

- *General Functions:* For general functions (provided by Zoho Deluge), square brackets **[]** are used. Arguments are passed as key-value pairs, enhancing clarity about which argument corresponds to which argument name:

```
generalFunction
[
    argName1: value1
    argName2: value2
]
```

- *Member Functions:* Member functions, linked to specific data types (like lists), use traditional parentheses **()**:

```
// Check if a list contains a particular item
listName.contains(item)
```

Data Types

In programming languages, a data type defines the kind of value a variable can hold, such as integers, floats, or strings. Deluge supports several data types, which help classify the nature of the information being handled.

Numbers: They can be categorized as whole numbers, integers, or those with decimal parts, known as floats. An example of an integer is 5, whereas 5.3 exemplifies a float.

Strings: Strings represent sequences of characters and are enclosed within double quotes. For instance, "Hello, World!" is a string.

Lists*:* Lists are ordered collections of items, and in Deluge, they can contain mixed data types. For example, a list might look like: [1, "apple", 3.14].

Booleans: These signify truth values and can either be True or False.

Maps*:* Maps are collections that store key-value pairs, allowing for efficient data retrieval using the key.

Operators

Operators in Deluge perform actions ranging from math to condition-checking. Here's an overview:

Arithmetic Operators*:* For simple math operations.

- +: Addition
- -: Subtraction
- *: Multiplication
- /: Division

Comparison Operators*:* For comparing values.

- ==: Equal to
- !=: Not equal to
- <: Less than
- >: Greater than
- <=: Less than or equal to

- >=: Greater than or equal to

Logical Operators*:* Checking conditions.

- &&: And
- ||: Or
- !: Not

Other Operators*:* Specialized operations.

+=, -=, *=, /=: Combine math with value assignment.

Basic Constructs

Like in other programming languages, constructs in Deluge offer a way to control the flow of the script, allowing for more complex operations.

Conditional Statements: These evaluate whether a statement is true or false and execute code based on that. This category is based on “if”, “else”, and “else if” keywords. For example:

```
if (x > 5) {
   // code to execute if x is greater than 5
} else if (x == 3) {
    // code to execute if x is equal to 3
} else {
    // code to execute if neither the first or the second condition is met
}
```

Loops: In many programming languages, a variety of loop constructs such as "for", "while", and "do-while" exist. However, in Deluge, the scenario is different. Deluge does not support traditional "for" or "while" loops. Instead, it exclusively offers the "for each" loop. This loop is specially tailored to iterate over collections, predominantly lists. Given the absence of other loop constructs in the language, the "for each" loop is used whenever iterative tasks are required. For example, suppose there's a need to iterate through a list of names and output them to the console:

```
nameCollection = {"Ella", "Mike", "Nora"};
for each  individual in nameCollection
{
    info individual; // Outputs value stored in "individual" to console
}
```

In each iteration of the loop, variable "individual" is assigned to a value from "nameCollection", in the order from left to right ("Ella", "Mike", "Nora").

2.6.5 Data Manipulation

Manipulating data is a standard feature of programming languages, including Deluge. For systems like Zoho that deal with a lot of data, it's especially important to know how to manipulate data properly.

Data Structures: Lists and Maps

Deluge uses two main structures for holding multiple pieces of data: lists and maps.

List: An ordered sequence of values. Each value is accessed by its position.

Map: Data is stored as key-value pairs. Each value can be accessed using its key.

Working with Data

For lists, values can be added or removed. A specific value can be retrieved using its position. Similarly to most other programming languages, the indexing starts at 0, so the first item is at position 0, the second is at position 1, and so on.

In maps, the key retrieves its corresponding value. Values can be changed, and new key-value pairs can be added.

Tools for Data

Specific operators in Deluge are meant for lists and maps. Functions designed to help manage data, such as arranging a list or searching for a key in a map, are also available.

Code Example:

```
// Using Lists
nums = {1, 2, 3}; // Define a list of numbers
val = nums.get(1); // Retrieves value "2" from the list (not "1", because of
zero-based indexing)
nums.add(4);  // Adds 4 at the end of the list

// Using Maps. Here "apple" and "banana" are keys, "5" and "10" are their
corresponding values
fruitCounts = {"apple": 5, "banana": 10}; // Define map of fruit counts
numApples = fruitCounts.get("apple");  // Gets apple count (5 in this case)
```

Basic data handling in Deluge was presented here. The code written for this project (presented in the practical part of the thesis) will use more advanced data manipulation functions and techniques. In many cases, it will describe what exactly the code is doing, or you can conclude this from the names of variables and functions. For other cases, it is suggested to check Deluge's online documentation.

2.7 Understanding API calls and their use in Deluge

2.7.1 Basics of API Calls

API (Application Programming Interface) communication is fundamental to present-day software development and linkage. They function as established rules, helping different software pieces work together easily.

You can imagine APIs like connectors in the vast realm of digital software. Much like couriers in real-world settings transport and relay packages between senders and receivers, API interactions enable data transmission between software components.

API interactions have become commonplace due to the expanded adoption of digital applications and systems in the modern world, which increased the need for reliable, secure and standardised linkage between them.

APIs often utilise HTTP (Hypertext Transfer Protocol) (or HTTPS) to enable communication between various systems. This protocol standardises how requests and responses are structured, ensuring consistent data exchange across the web.

Here are the types of requests supported by API:

GET: Used to collect data from a chosen location - a server. Upon a GET request, the server replies with the requested information to the user. For instance, accessing a website often involves a GET request to fetch its details.

POST: Unlike GET, POST sends data for processing to a server. This could involve transmitting form details or modifying server data. Upon data reception, the server processes it and replies if needed.

Other methods: PUT (modifying current data), DELETE (erasing data), ...

In API interactions, headers play a significant role. They contain the request's metadata, like the type of content transmitted and the requestor's authorisation data.

In the context of APIs, RESTful or Representational State Transfer is an architectural style for designing networked applications, which has become the standard for modern web communication. It uses a stateless, client-server communication model, making it widely used because of its simplicity and scalability, especially when contrasted with alternatives like SOAP.

To understand why API is so predominant these days and to understand it better, let's compare it to another similar protocol - SOAP (Simple Object Access Protocol). Both API calls (specifically RESTful APIs) and SOAP are designed for communication between software components, but they have distinct differences:

Simplicity: RESTful APIs are generally simpler and more flexible than SOAP, which requires a rigid structure.

Data Format: SOAP strictly uses XML, whereas RESTful APIs can also utilize JSON, which is often more user-friendly.

Standards: SOAP has extensive standards, making it suitable for complex tasks. However, this can be excessive for basic operations, where the lighter RESTful APIs are more appropriate.

Flexibility: APIs provide more flexible data manipulation than SOAP's standardized operations.

Due to its robustness and support for complex operations, SOAP is nowadays only used in some legacy systems, especially in enterprise settings and industries like finance and healthcare. In most other cases, API calls' practical advantages and adaptability over alternatives like SOAP have made them a go-to tool in today's digital platforms for communication between software components.

2.7.2 Deluge and API Interactions

Deluge, Zoho's scripting tool, facilitates various operations and has a robust capability for API interactions. This feature is not confined to Zoho's application environment alone but also extends to third-party applications and services.

Technically speaking, Deluge simplifies the process of making API calls by providing its functions. These functions can be used to connect to Zoho’s applications or other network applications that provide their APIs. Using HTTP as the underlying protocol, Deluge scripts can send requests to various external services and handle responses.

For instance, if a Deluge script wants to fetch customer details from an external database through an API, a common method used is GET. This method is employed to retrieve data from a specific resource.

```
response = invokeurl
[
    url: "https://api.externaldatabase.com/customers/12345"
    type: GET
];
```

On the other hand, if the script needs to send data, e.g., post customer details to another system, the POST method can be used.

```
// Create a map that holds the values of what to post to the API
customer_info = Map();
customer_info.put("customer_name", "Shawn");

// Format the data so that it can be posted to the API
data  = Map();
data.put("JSONString", customer_info);

// Supply the URL and parameters to the invoke the POST task
response = invokeurl
[
    url: "https://api.externaldatabase.com/customers"
    type: POST
    parameters: data
];
```

Upon executing these API calls, the external services provide responses, often structured in JSON or XML formats. Deluge comes equipped with functions that parse these data structures so the script can seamlessly process the received information:

```
// Get value by "status" key if the response is in JSON string format
info response.getJson("status");
```

Security is a critical aspect of almost any API communication. Deluge addresses this issue by supporting various authentication methods. Often, API calls require API keys or OAuth tokens to ensure the calling script has the proper permissions. To support authentication in the `invokeurl` function, a new »connection« can be created in Zoho's settings. Then, the connection's name can be provided as a »connection« argument to the function to use the authentication method set up in the settings beforehand.

```
response = invokeurl
[
    url: "https://api.externaldatabase.com/data_requiring_autentication/12345"
    type: GET
    connection:"example_conn"
];
```

For a Deluge script to communicate efficiently with an API, the developer needs a good understanding of the API's documentation. Knowledge about specific endpoints, the structure of the data being exchanged, and any required headers is crucial. While Deluge simplifies the scripting part, the external service's documentation offers insight into creating effective API calls.

2.8 Agile development strategy

2.8.1 Agile Development, Iterative Development, and Incremental Development

Agile development is an umbrella term that includes iterative, incremental, and some other development styles. These practices use flexibility, adaptability, and continuous improvement in the software development process. These approaches are opposed to traditional waterfall models. One of the most significant differences between agile and waterfall development is that waterfall models (as opposed to agile) emphasise much pre-planning for the project. This stage includes a detailed sequence of sub-stages in which the project will be executed. This often leads to inflexibility and delays when changes are needed.

Agile development is an overarching approach to software development that promotes delivering small product segments through collaborative teams, including employees from different areas within an organization. Agile frameworks, such as Scrum and Kanban, advocate for frequent reassessment of project goals, allowing teams to adapt quickly to changing requirements. Agile prioritizes customer satisfaction, embraces change, and develops a collaborative environment among developers and stakeholders.

Iterative development focuses on developing software through iterations. Each iteration involves planning, designing, coding, and testing. This method allows for regular re-assessment and adaptation, ensuring the software evolves based on user feedback and testing outcomes. Breaking down the project into smaller parts allows developers to manage risks better and incorporate new insights without disrupting the entire project.

Incremental development involves building software incrementally, adding new features in small, manageable pieces. Each stage (increment) builds upon the previous ones, progressively improving the product's functionality. This method allows for early delivery of a usable product (without all of the features), which can be beneficial for receiving early feedback and making necessary adjustments based on that (Dingsøyr et al., 2012).

This kind of development also allows for adapting to new requirements of the project as they become clearer »on the go« along the development of the project

2.8.2 Single developer versus a team of developers

Another valuable aspect to consider about software development is the difference between the development of a single developer and a team of developers.

Working as a single developer versus a team differs significantly in software development. A single developer can benefit from simpler (and less) communication and quick decision-making (since he/she doesn't rely on other developers), often leading to a more consistent

codebase. However, this approach struggles with larger projects that require diverse expertise and can become overwhelming.

On the other hand, teams bring together various skill sets, creating innovation through collaboration. This diversity allows for handling more complex projects efficiently, although it introduces challenges in maintaining consistent communication and managing potential conflicts. Ultimately, the choice between a single developer and a team depends on the project's complexity and requirements. If the project is big, a team of developers is usually needed. However, one also has to consider the additional complexity of communication within a team (which contributes to additional delays), which is not the case for a single developer.

A single developer might find Agile methodologies very useful. They allow for managing complexity by breaking tasks into smaller, more manageable units. The iterative and incremental approaches enable a lone developer to make continuous progress, handle changes efficiently, and maintain a flexible schedule.

A team of developers working on Agile projects can leverage the collective expertise and collaborative approach that Agile promotes. Teams can distribute tasks according to the strengths of each individual, communicate regularly through daily stand-ups (short meetings where each team member presents what he/she is currently working on), and conduct sprint reviews to ensure alignment with project goals. Agile frameworks like Scrum provide structure to team-based product development, ensuring that all members are synchronized and contributing towards incremental product/service improvement.

Overall, Agile development methodologies offer all-encompassing frameworks for managing software projects. Whether applied by a single developer or a team, these approaches increase flexibility, encourage continuous improvement, and deliver high-quality software that meets user needs. Because of these advantages, Agile development is significantly more widespread than the traditional waterfall models (especially in modern companies).

3 RESEARCH METHODOLOGY

3.1 Research approach

My research was divided into two main tasks. The first one was finding potential improvements in the current supply chain. The second one presented a solution to a major problem in the supply chain—the lack of automation in the company’s order fulfilment process. In this section, I will describe the research approach for both tasks.

First, I considered the Chinese supply chain. The company Ubiquity Robotics is relatively new to the market, so it didn't yet have a mapping of that supply chain, which meant I

couldn't rely on it. That's why I had to analyse not only the future but also the current supply chain state. I did that with the help of the general manager and administrative assistant because they know the most about it.

I used value stream mapping to represent the current and future states. First, the VSM map of the current state was drawn, and then the problems were identified. Then, the map of the potential future state was drawn, which addressed the problems of the current state.

To move from the current state to the future state, we used the A3 technique for problem-solving.

To draw VSM diagrams of both tasks, I used a free online tool, »diagrams.net«, which has all the necessary functionality and is comparable to similar online tools that must be paid for.

The order fulfilment process was automated using the Deluge scripting language in Zoho's low-code programming environment.

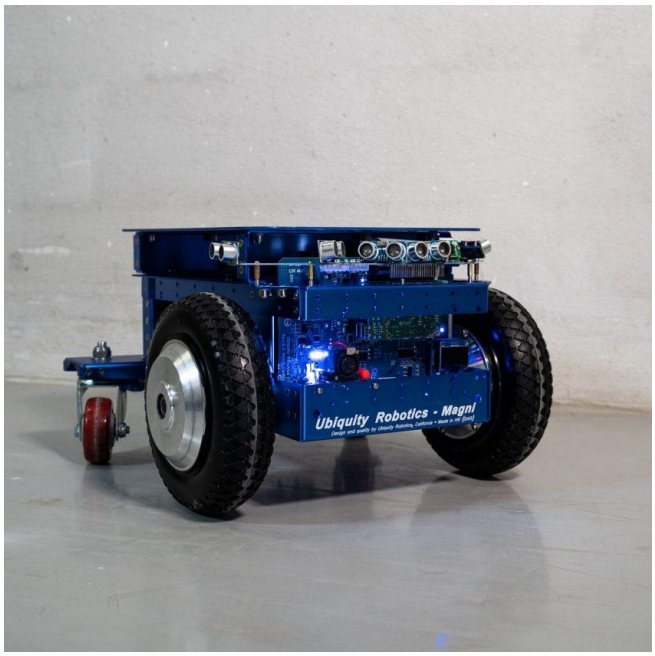
3.2 General information about the company

Ubiquity Robotics is a US-based manufacturer of general-purpose mobile robots. The company owner is David Crawley. It was officially founded in 2017, but the work on its base robot, Magni, started a few years earlier. Soon after its foundation, the owner moved to Slovenia and created a subsidiary (LLC), which is now the company's main part. Research and development are now conducted in Slovenia.

Most of the company's revenue is currently generated with its general-purpose robot, Magni (Figure 6). However, there are also a few specific-purpose robots being developed. One of them is the Conveyorbot (Figure 7), a robot that moves things around in a warehouse or any other place where things have to be moved around. The robot navigates by relying on floor stickers detected by the camera, containing codes similar to QR. There is also a robot similar to this one in functionality called EZ-Map, but it navigates using a sensor called LIDAR, which creates a 2D scan of its environment. Software is then used to navigate this scanned environment on user-defined routes, defined by placing goals on the web user interface. A recent project was a merge of these two navigation software into one software, which has benefits of both.

Other specific-purpose robots being developed are a person-following robot, a disinfection robot, a cocktail waiter robot, and others.

Figure 6: Magni



Source: Ubiquity Robotics (2024)

Figure 7: Conveyorbot

Source: Ubiquity Robotics (2024)

An important revenue stream of the company is also custom robotics projects developed for other companies.

The company also makes SD-card images for the Raspberry Pi computer with a pre-installed operating system, Linux, and a framework for robotics development called »ROS« (which we use for development). These images are freely available on the company's webpage. They are widely used by hobbyists, students, and other robotics companies.

3.3 Data collection

To generate current and future VSM maps of the supply chain, I collaborated with coworkers (general manager and administrative assistant) to gather the necessary data. We had multiple meetings about the topic to determine the project's details and scope. Most of the discussions were between me and the administrative assistant since the general manager is usually busy with other work. Also, because the administrative assistant is dealing with the order fulfilment process (currently with mostly manual work), so she knows the most about it – what exactly needs to be automated, and in what way – based on how she is currently doing it.

The meetings/discussions had two main stages. The first stage involved analysing the current state of the supply chain (and the order fulfilment process specifically) in more detail to arrive at its accurate representation. The second stage concerns what parts of the "current state" to automate and how to automate them to arrive at the desired "future state" of the supply chain.

When automating the order fulfilment process, a spreadsheet was generated containing information about the preferred warehouse to ship from to each country. This spreadsheet was queried using the automated warehouse selection procedure when processing orders. To gather this information, I considered distances between warehouses and each country and the associated shipping costs.

The general manager and the administrative assistant did other work and research related to this project that I was not/will not be collaborating on, for example, searching for the actual order fulfilment centre/warehouse that the company will use in Canada and for other parts of the supply chain there.

4 RESULTS

For the solution to our problem, we followed the A3 problem-solving technique. The issue and background were described in the introduction. The following steps (current condition, goal, root cause analysis, target (future) condition, countermeasures, implementation and cost analysis) will be described in this section. The last two steps (test and follow-up) will

be omitted because we haven't reached this stage of development yet (the solution is not yet fully implemented and operational). Thus, we can't test its performance and do a follow-up.

The current and future conditions were described/represented with the VSM technique.

4.1 Current supply chain state

In this section, I will describe the VSM map of the company's current supply chain state and the problems we discovered while and after drawing the map.

In our current state map (Figure 8), we focused less on the production process and more on logistics—warehousing and transport—because those are the parts that need the most improvement.

4.1.1 Material flow

I will start by describing the material flow. Our robots are assembled in China in a factory named "Kindle". Part of the supplies that the factory needs to create our robots are ordered by the company (Ubiquity Robotics), and the other part is ordered by the factory itself. The company orders wheels and printed circuit boards (PCBs), and the factory orders sheet metal and various "China-supplied peripherals" themselves. PCBs are ordered from two different suppliers - Nikolas and Athena. Usually, they are ordered once from one PCB supplier and one from another. This enables the company to negotiate the price. Having two suppliers also reduces the risk of being dependent on only one supplier, which might lead to problems if this supplier goes bankrupt or can't supply the PCBs anymore for some other reason. All other goods are supplied in a pull way, except PCBs. Those are supplied in a push way because there is a significant reduction in price per unit when ordering a larger number of PCBs at once.

The assembled robots are sent to a Hong Kong warehouse and fulfilment centre named ZhenHub. The last assembly part is done, which involves inserting the SD card with the installed software into the robot. The SD cards are obtained from a US supplier when needed (pull). From ZhenHub, the robots are shipped to customers worldwide via aeroplane shipment.

4.1.2 Information flow

Now, I will describe the current state of information flow.

Zoho is a software-as-a-service solution used to manage the information flow (and many other parts of the business).

Customers can order from the online store or by direct mail to the sales department. For our webpage, we currently use WordPress with Shopify plugin for e-commerce purposes.

If the order is made by direct mail to the sales department, the sales department first negotiates with the customer about the price and then issues an estimate. Once the estimate is approved, the retainer invoice is sent to the customer. After payment is received, an email with order information is sent to "Deepak" (our external co-worker who is in charge of order processing and shipment tracking), who manually enters order details into the ZhenHub interface, where they create shipping labels and commercial invoice and package slip (a list of what is contained inside the shipment). After that, the shipment is dispatched from ZhenHub to the customer. Meanwhile, the sales department issues an invoice, applies payment and emails the invoice to the customer. The tracking number is sent to the customer manually from ZhenHub as soon as it becomes available.

If the order is made on the online store, the process is a bit more automated. After the order is made, a sales order is generated in ZhenHub automatically, but needing Deepak's approval. From this point on, the order is processed in the same way as if it was performed via email.

In both order options, Deepak also performs monitoring of shipment while it is in transit from the warehouse to the customer.

There is also a "Control tower", which currently only includes the administrative assistant. She monitors the inventories of suppliers and our warehouse (ZhenHub). If the inventory of our assembled robots in ZhenHub goes below 5, she contacts our suppliers to create new robots.

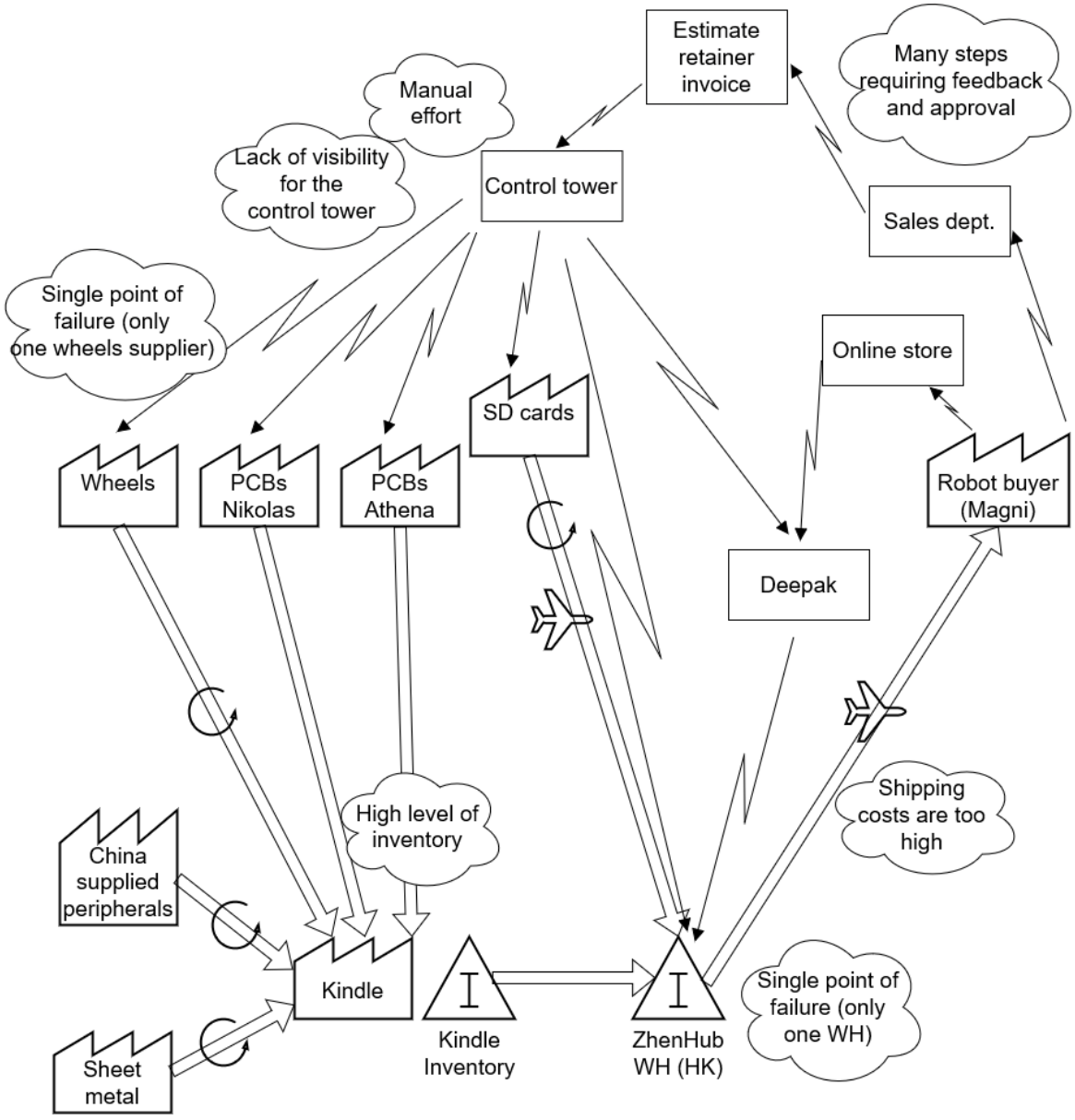
4.1.3 Discovered problems

During and after drawing the current state map, we discovered several problems presented in the VSM in cloud shapes. The following chapter will address these problems, where the future state map will be presented. It will contain improvements to the current state map to remove those problems.

The biggest problem we currently face is the high shipping cost from the ZhenHub warehouse (located in Hong Kong) to customers worldwide. The cost is so high because we ship by aeroplane (a high-speed but costly way of transport) from only one warehouse worldwide. Most orders (50%) are made from the United States, on the other side of the globe from Hong Kong. This is a severe logistics problem.

Another set of problems is vulnerability due to a single point of failure (SPOF). We only have one warehouse and only one supplier of wheels for the robot.

Figure 8: Current supply chain state



Source: Own work.

A problem is also the fact that many steps in the supply chain require feedback and approval, creating time delays. Also, a lot of manual effort (administrative work) is required through the supply chain, like manually entering data into Zoho web forms. In other words, the supply chain should be more automated.

A problem is also a high level of inventory. If we have inventory in the warehouse for over a month, warehouse expenses are high. It would be ideal if we restocked in the same month; in that case, only a flat rate would have to be paid.

There is also a problem with the Control tower: its lack of visibility over the suppliers. The suppliers always have an approximate idea of the number of supplies, but to get the exact

number, a slow process needs to be performed. First, email a supplier, who has to send someone into the warehouse to check the exact level of supplies we enquired about. After that, this information is finally sent back to the Control tower.

4.2 Goal and root cause analysis

The main goal is a reduction of supply chain costs. The other two goals are the reduction of SC Order lead time (time from customer order received to customer order delivered) and risks originating from a SPOF.

Figure 10 presents root cause analysis of high costs and Figure 9 present the root cause analysis of high lead time and vulnerability due to SPOF. We have found several root causes, some of which are not solvable in the scope of my thesis. I will elaborate on each root cause and present a way of preventing its repetition in the future in the continuation of this section.

We only have a single warehouse for finished products: This root cause will be resolved by adding one or multiple new warehouses to the supply chain.

High shipping cost from suppliers to Kindle: This root problem can't be solved because it is essential for the supply chain. However, these costs could be reduced if there were more alternative assembly centres in the supply chain because they would shift some of the load to them, and thus, Kindle would need less material from its suppliers.

High shipping cost from Kindle to ZhenHub: This root cause is also essential for the supply chain, and the cost could also be reduced if the load is shifted from ZhenHub to other warehouses.

The pull method is not properly established: Establishing it will take quite some effort, so it is out of the scope of the thesis.

Most orders are made by people who experiment with the robot: This is not a problem by itself, but it causes a problem that most robots are ordered individually, not in a bundle. This will be resolved when (if) more Conveyorbots are ordered compared to Magnis because Conveyorbots are meant to operate mainly in large warehouses where more than one robot will be needed.

Orders made through the sales department need to be entered manually into Zoho: This problem is not solvable because we can't just conclude on the order automatically based on a conversation between the sales dept. and client.

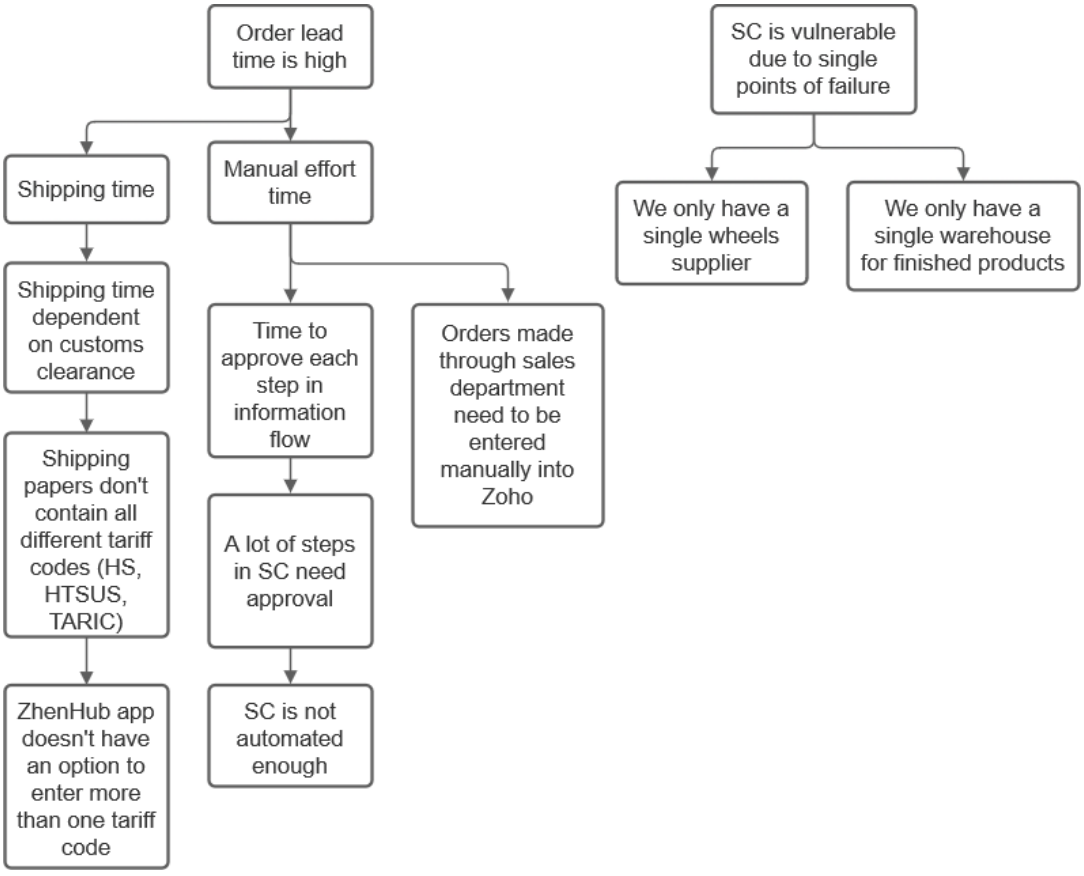
ZhenHub has no open API or integration with Zoho: This problem can only be solved by ZhenHub itself. They said they are working on it and will solve this in a few months.

SC is not automated enough: This problem will be dealt with by increasing the automation of the information flow by using additional Zoho features. The automation of information flow, specifically the automation of the order fulfilment process, is the most significant part of the practical section of this thesis.

We only have a single wheel's supplier: This problem can't be solved right now because we don't know any other suppliers which would have the kind of wheels we need.

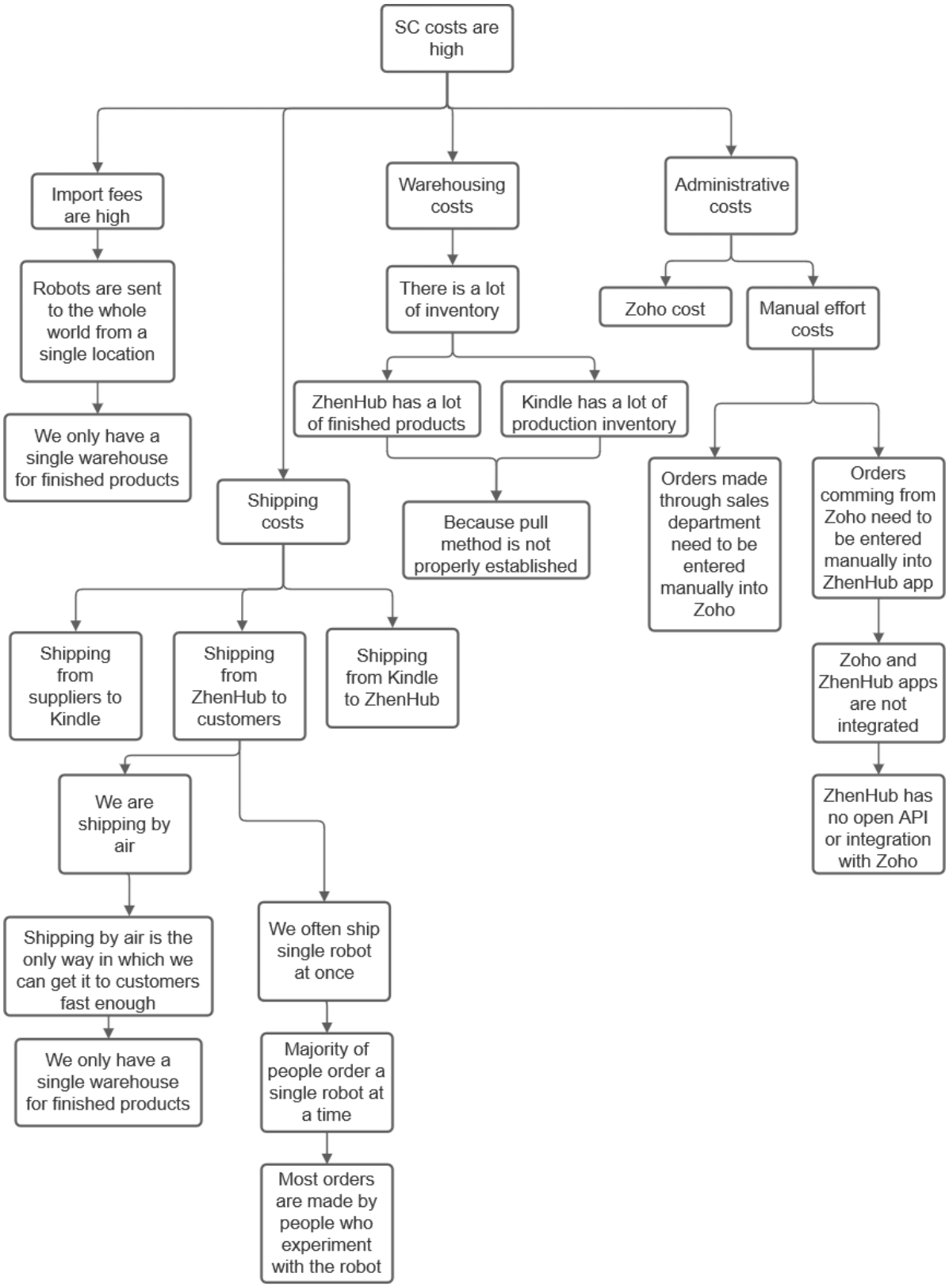
ZhenHub app doesn't have an option to enter more than one tariff code: We haven't researched what would be the best solution for this yet; this is out of scope for this thesis.

Figure 9: Root cause analysis of high lead time and vulnerability due to SPOFs



Source: Own work.

Figure 10: Root cause analysis of high cost



Source: Own work.

4.3 Suggestion of the future SC state

Now, the planned future state map will be presented. The shapes on the map are the same as those on the current state map.

4.3.1 Material flow

As evident from Figure 11, the material flow is similar to the current state map. All parts of the material flow will be the same until the Kindle inventory. From that inventory, our products are sent not into one but into two different warehouses, one of which is the same as before - ZhenHub in Hong Kong and another one will be a warehouse in Canada we haven't used before. Having multiple warehouses reduces the single point of failure risk, which occurs if there is only one warehouse. The other and even more important advantage is that the shipping costs are lower when goods are distributed worldwide from multiple warehouses. In this way, two of the problems (clouds) drawn in the current state map are eliminated - "shipping costs are too high" and "vulnerability due to a single point of failure in the warehouse"

The new warehouse will be located in Canada because it is near the United States, and, as aforementioned, the majority of the orders are received from the US. Also, the import costs are lower than if we had a warehouse in the US (they are around 5%).

Shipments to the new Canada warehouse are planned, and they will be made by ships, which are much cheaper means of transport than aeroplanes. From that warehouse, our products will be distributed to the US by trucks, which are also cheap ways of shipping.

Another warehouse in Europe might also be obtained in the future to reduce shipping costs further, but this is out of the scope of this report.

4.3.2 Information flow

Unlike material flow, the information flow has major changes in the future state compared with the current state. The cloud software Zoho is intended to be used in a more advanced way than it currently is.

In Figure 11, the desired future state of the information flow is presented.

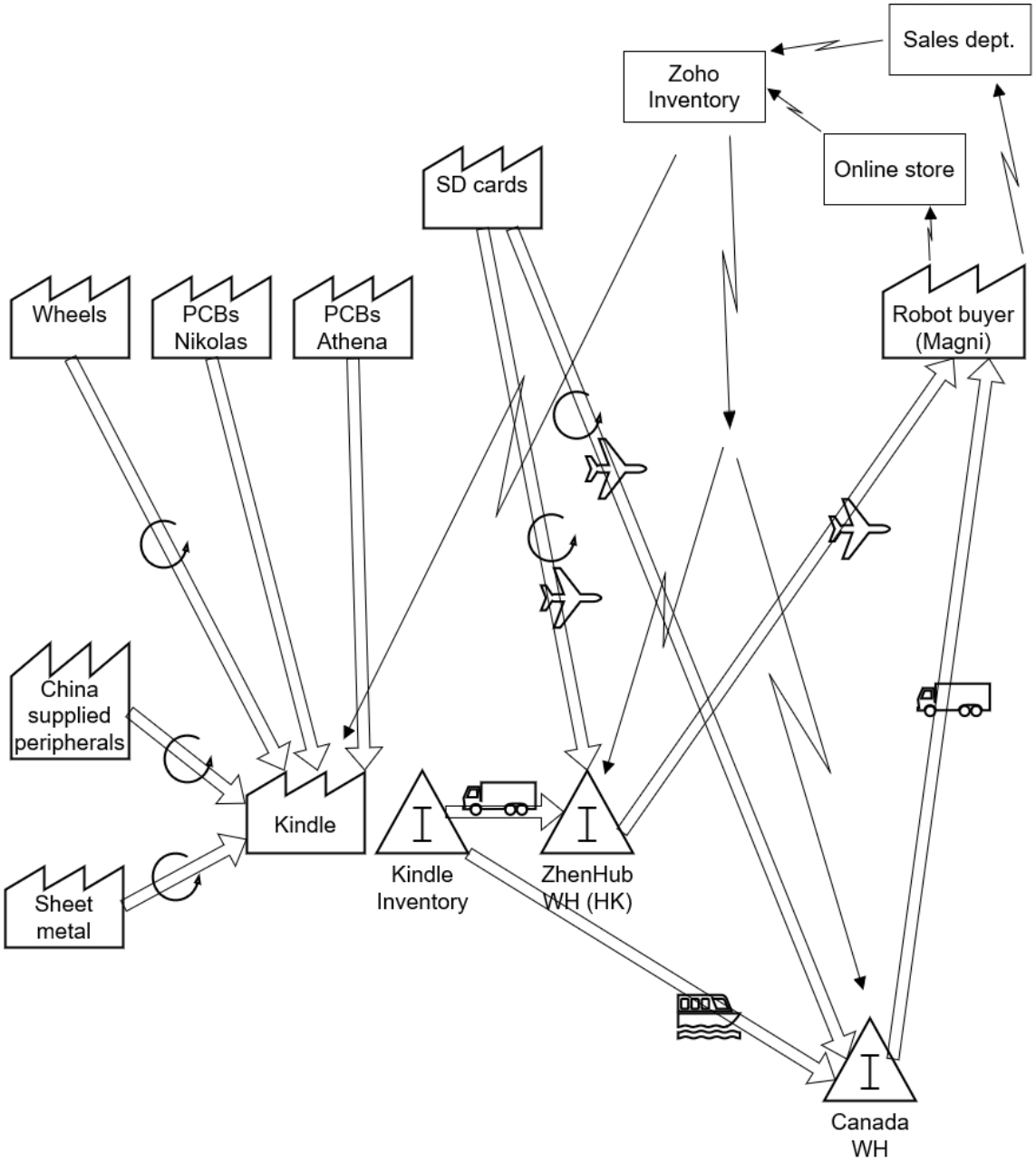
As seen from the figure, we would first want to achieve that ordering from our online store and the contact with the sales department would both lead to a sales order generated in Zoho Inventory. In the case of ordering through the sales department, manual entering of the buyer's information into Zoho Inventory's web form will still be required. However, if the order would be placed in a web store, this process could be done automatically.

After creating the sales order, a Deluge script will decide which warehouse to ship the product from based on the shipping address. This script will also keep track of the inventory levels and update them according to what items were ordered in the sales order.

Using Zoho's advanced features would reduce a significant amount of work currently done manually. This would significantly reduce the need for manual effort and required working time to manage the supply chain.

The main goal of the practical part of this thesis is to present Zoho's low-code software development platform, which uses the Deluge programming language to achieve a high level of automation in the order fulfilment process.

Figure 11: Future supply chain state



Source: Own work.

4.4 Analysis of the developed code for the order fulfilment process

In this section, the code developed for the automation of our order fulfilment process will be described. The code is not fully functioning yet; for example, it is not yet connected to ZhenHub's warehouse management system to which the orders should be forwarded.

First, the code for handling orders from WordPress' web store using WooCommerce will be described. The code is made as a Zoho Inventory's "Custom function", which is Zoho's way of dealing with their objects, like "Sales orders". The custom function can then be referenced from "Workflow Rules", which can trigger the functions automatically, for example, when a new Sales order is received from a web store.

At the end of the section, a Zoho Creator GUI application will also be presented shortly, intended for warehouse packers who dispatch our products.

The two pieces of software are meant to be used in combination. After the packages corresponding to appropriate warehouses are created and a sales order is processed, each warehouse should dispatch its corresponding packages. After that, the packer should enter the package information in the GUI application and submit the information to be processed by Zoho Inventory's information system, which also sends an email to the customer with the shipment information.

The software development style was the closest to Incremental development. I started with minimal functionality at first and then incrementally added the additional required complexity, which was crystallised by the previous increments. In some cases, it turned out that I did too little planning, so I needed quite some code refactoring after a new problem (or a required feature) became apparent.

4.4.1 Custom function for processing new orders

Discussion

The code is responsible for processing sales orders after a customer creates them in our web store. It operates within the context of the Zoho Inventory platform, interfacing with its API to manage sales orders based on available stock across multiple warehouses.

In essence, the function carries out a series of steps to determine which warehouse a specific sales order should be fulfilled based on product availability and shipping costs. First, it identifies the customer's country, determining which warehouse is the preferred location for shipping. The process includes checking the inventory for items already packed from that

sales order, updating those not packed yet to a "Not selected" warehouse status, and adjusting the sales order based on the stock levels in the primary and secondary warehouses.

The function prioritizes the preferred warehouse for fulfilment. However, if the required quantity is unavailable in the preferred warehouse, the function considers a secondary warehouse as the alternative source. If neither warehouse has sufficient stock, the function marks the item as "Out of stock". On Sales orders containing “Out of stock” items after being processed, the Custom function can later be manually triggered again when new stock arrives at the warehouse(s).

Description of the algorithm

Individual parts of the algorithm will be presented with pseudocode here. The actual code is located in Appendix 1. Each block of the pseudocode follows the previous one.

1. Custom function declaration

```
void process_sales_order(Map salesorder, Map organization, Map user)
```

`void` means that the function doesn't return anything. `process_sales_order` is the name of the function, and there are three input arguments of type `Map`:

- `salesorder` is a map containing information about the new Sales order that is going to be processed
- `organization` is a map containing information about the organization (in this case Ubiquity Robotics)
- `user` is a map containing information about the logged-in user. We are not going to use it in this function.

2. Variable initialization

- Dictionaries for information about individual SKUs (like package dimensions) and mapping of warehouse names to their corresponding IDs are defined.
- Details such as sales order ID, organisation ID, customer ID, and shipping address country code are extracted from the Sales order and Organization objects.

3. Preferred warehouse selection

This code fetches records (rows) from a Zoho Sheet and checks them to determine the preferred warehouse based on the country in a sales order's shipping address.

When the records from the sheet are fetched, they are returned as a list of maps. Each map in the spreadsheet contains information about an individual row. The keys of these maps

represent column headers, and values represent the data contained in the corresponding cell of that row.

The code assumes that each row in the Zoho Sheet has column headers representing warehouse names and cell values representing countries. When deciding from which warehouse to ship the items in the sales order, the preferred warehouse is the one that has the buyer's country in its column. When the code finds the country in a row that matches the one from the sales order, it takes the column header of that cell as the preferred warehouse. If the country is not found in any of the columns, the default preferred warehouse is the warehouse in China (ZhenHub). Currently, the only other warehouse is Canada's warehouse, so the spreadsheet only has one column related to that warehouse, containing the countries to which to ship if the buyer is from that country.

After determining the preferred warehouse, the code assigns the secondary warehouse. An error message is logged if the preferred warehouse is neither of the two known warehouses. This could happen if a warehouse in the sheet is not recognized (if it's neither of the two warehouses).

The preferred warehouse name is extracted from the records with the following pseudo-code:

```
For each record in response.get("records")
    If the record contains a value that matches the country
        For each key in the record's keys
            If the value associated with the key matches the country
                Set preferredWarehouseName to the key
                Break out of the inner loop
        Break out of the outer loop
```

4. Packed items retrieval and update

Sales orders can be converted into one or more "packages", which are groups of items that will be sent as a package from a particular fulfilment centre. This is done later on in the function.

This step is needed to see if some items in the currently processed sales order have already been packed. If so, these items will not be converted to package(s) again later. This situation can happen if the order has already been processed once before, but due to the lack of inventory, some of the items haven't been sent to the client yet. In this case, the order can be processed again with the same function. This part of the code is needed to determine which orders have already been processed/packed to the function in the previous call(s).

First, a call is made to Zoho Inventory's API to obtain a list of packages already created from the sales order being processed. These packages are then iterated, and in each iteration, details of each are retrieved with an API call.

Each sales order contains the field “line_items,” which lists the individual items ordered in the sales order. The associated field “quantity” tells how many items of this kind have been ordered. The field “line_item_id” contains a unique ID for each line item.

Each package has a “line_items” field as well, which lists all the items within that package (that are to be sent to the client). Each line item in the package contains a field "so_line_item_id," which contains the same value as the “line_item_id” from the corresponding sales order.

“item_id” is the id of an item in the Zoho Inventory. This id is unique for every item the organization has in the Zoho Inventory. “line_item_id” is a unique id corresponding to each Line item in a particular sales order. It should not be confused with the item_id.

The `sentItems` map is structured to have "item_id" as its keys and another map as its values. This inner map has "wh_id" (warehouse id - unique for each warehouse) as keys and the packed quantities as values. The `sentItemIds` list stores the IDs of the line items in the sales order that have already been packed.

The code uses two nested loops: the outer loop goes through each package in the sales order, and the inner loop goes through each line item within those packages. It updates the `sentItems` and `sentItemIds` as it iterates through these line items.

The following pseudo-code determines the quantity of each item, identified by "item_id", that has been packed from each warehouse, identified by "wh_id".

```
Fetch all packages associated with the sales order
Initialize sentItems as an empty map
Initialize sentItemIds as an empty list

For each package in the fetched packages
    Fetch details of the individual package
    For each line item in the package details
        Add the line item's ID to sentItemIds
        Get item_id and wh_id from the line item

        If item_id exists in sentItems
            Get the map of warehouses and quantities for this item
            If wh_id exists in the map
                Update the quantity for this warehouse in sentItems
            Else
                Add a new entry for this warehouse with the current quantity
            Update the map in sentItems
        Else
            Create a new map for warehouses and quantities
            Add an entry for the current warehouse with the current quantity
            Add the new map to sentItems
```

5. Update unpacked items

This code snippet continues to work with the sales order and its line items, focusing on those items that haven't been fully packed yet. As in the previous part of the code, this one is also used for subsequent calls to the function (after determining which items have already been packed during the last call(s) to the function). It updates the quantities and warehouse details for each line item and then sends a request to update the sales order in the Zoho Inventory with the modified line items. This updating is necessary so that the Zoho Inventory's stock updates.

The code first updates the sales order line items based on their packed status. For items already packed, the quantities are adjusted by subtracting the already packed quantity from its quantity in the sales order. For items not packed or only partially packed, it sets the warehouse as "Not selected" to indicate that the appropriate warehouse from which to ship will need to be selected in the following step. Finally, it sends a PUT request to Zoho Inventory's database to update the sales order with the adjusted line items.

This code ensures that all of the line items' packed statuses and quantities are correctly represented in Zoho's database before updating the sales order.

```
Initialize list lineItemsUpdated1

For each lineItem in salesorder.line_items:
    Initialize lineItemUpdated with line_item_id
    If item_id in sentItems and warehouse_id in sentItems[item_id] and
line_item_id in sentItemIds:
        Calculate remaining packed quantity num_packed_remaining
        Update packed quantity:
            If num_packed_remaining > lineItem.quantity:
                Decrease num_packed_remaining by lineItem.quantity
                Update sentItems for item_id and warehouse_id
            Else if num_packed_remaining == lineItem.quantity:
                Remove warehouse_id from sentItems for item_id
            Else:
                Set lineItemUpdated.quantity to num_packed_remaining
                Remove warehouse_id from sentItems for item_id
                Create new lineItemUpdated2 for remaining quantity
                Set warehouse_id of lineItemUpdated2 to "Not selected"
                Add lineItemUpdated2 to lineItemsUpdated1
    Else:
        Set lineItemUpdated.warehouse_id to "Not selected"

    Add lineItemUpdated to lineItemsUpdated1

Set params with updated line_items lineItemsUpdated1
Send PUT request to update sales order in Zoho Inventory
```

6. Sales order stock update

This is the main (and biggest) part of the code. It manages and updates the line items in the sales order, specifically regarding stock and warehouse assignments. It does this in the following steps:

1. Fetches the sales order to get up-to-date stock information.
2. Retrieves stock quantities for each item in the order across all warehouses.
3. For each line item in the sales order, it evaluates and updates the warehouse to fulfil the order based on stock availability. If an item isn't fully stocked in one warehouse, it tries to fulfil the remainder from a secondary warehouse. If neither warehouse can fulfil the order, it sets the line item to "Out of Stock."
4. The updated line items are then pushed back to the sales order, and Zoho’s database is updated with the new information about this sales order.

```
Fetch sales order and store line items in lineItemsNew1

Initialize stock map
For each lineItem in lineItemsNew1:
    If item_id not yet in stock:
        Fetch stock details for item_id
        Store stock availability by warehouse in stock map

For each lineItem in lineItemsNew1:
    Initialize lineItemUpdated with line_item_id
    If warehouse_id is selected, skip to next lineItem
    Set tax details based on country code
    Add package details if available

    Determine stock and assign warehouse:
        Fetch stock for item_id from preferred and secondary warehouses
        If enough stock in preferred warehouse:
            Deduct stock and set warehouse_id to preferred
        Else:
            Allocate available stock from preferred warehouse
            Determine remaining quantity to be fulfilled from secondary
warehouse
            If sufficient stock in secondary:
                Set remaining quantity and update stock
            Else:
                Allocate all available stock from secondary
                Determine remaining quantity and set as "Out of Stock"

    Add updated lineItems to lineItemsUpdated list

Update sales order in Zoho database with modified line items
```

7. Package generation

This is the last step, which processes the updated sales order to generate packages for each warehouse. For each line item in the sales order, properties such as total weight, total price, and number of pieces are calculated, and this information (in combination with other information from the sales order) is used to generate packages. Once the packages are created, they are published to the Zoho Inventory database using their API. Each package's total price is calculated, so the invoice can be later generated out of it when the package is sent out of the warehouse. This happens when the packer submits the information about the sent package in our GUI application (which will be presented in the following section).

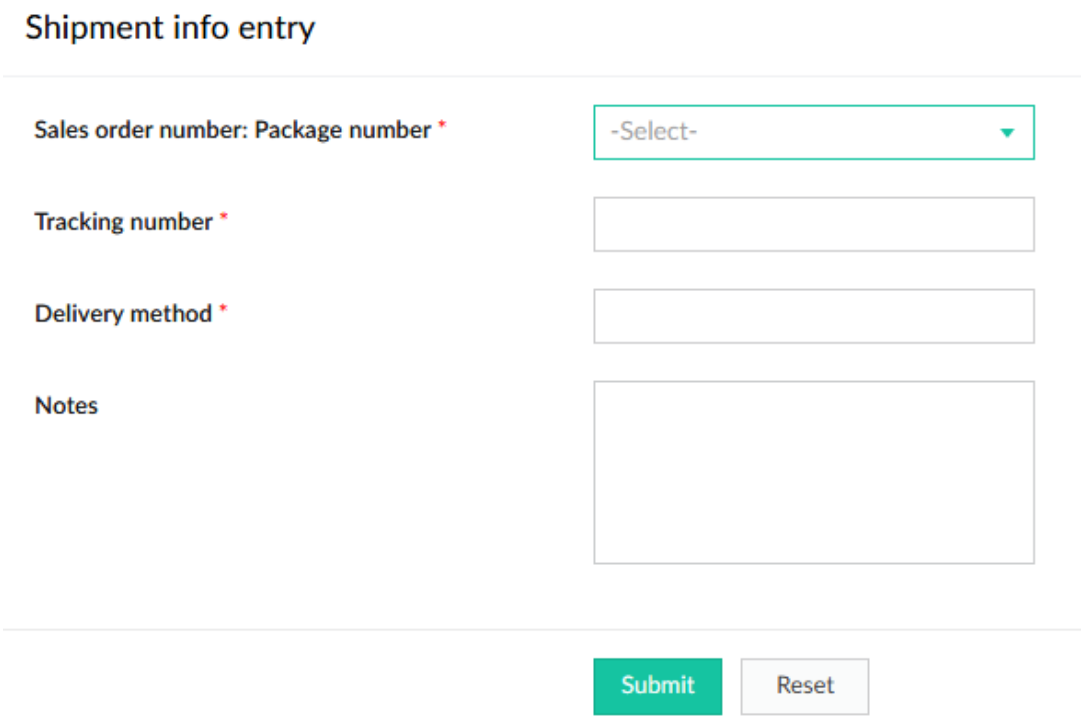
```
Fetch sales order details from Zoho Inventory

Initialize packageDetails map for each warehouse

For each lineItem in sales order:
    If lineItem is already sent, skip
    Extract line item details and custom fields
    Add line item to corresponding warehouse in packageDetails

For each warehouse in packageDetails:
    If warehouse is 'Out of stock' or has no items, skip
    Initialize totals for weight, price, and number of pieces

    For each lineItem in warehouse:
        Calculate totals for weight, price, and pieces from custom fields
        Clean up custom fields from lineItem

    Construct package details with totals and date
    Post package to Zoho Inventory
```

4.4.2 Zoho Creator application for packers in the warehouse

In the theoretical chapter, I touched upon the importance of reducing the order picking/packing time. This application (Figure 12) is an attempt to do so to some extent. It provides a simple GUI for warehouse packers who dispatch the ordered products. With this application, marking which order is about to be shipped takes less time.

On startup, the application fetches Zoho Inventory's packages, which haven't yet shipped. Packages contain information about what items from a particular Sales order should be packed together in one specific warehouse to be dispatched. A dropdown list on the GUI is then populated so that the packer can select the package which will be dispatched. The following figure shows the GUI.

Figure 12: Shipment information entry application



Source: Own work.

The packer first selects a package by its Sales order number and Package number on the »Sales order number: Package number« dropdown list. Then, he enters its related information - tracking number and delivery method - and optionally adds additional notes regarding the package.

When the "Submit" button is clicked, a "Shipment" (which is also a Zoho object) is generated out of the Package. Shipments represent packages that are in transit to the buyer. This also marks the package as "shipped," so it will not be detected by the application anymore. After that, an Invoice for the user, which is a Zoho Book object, is generated. It is created by invoking the Zoho Book's API and also sending the related sales order ID.

5 DISCUSSION

5.1 Interpretation of main findings and contributions

Many potential improvements to the supply chain have been identified. The results of the research supported the decision that the company was already planning to make - to use

another warehouse in Canada (besides the current one in China). To this new warehouse, the products are transported by ships. From it, they are distributed to clients by trucks.

Many other problems have been identified, but we didn't attempt to solve them because they weren't currently resolvable or because they were out of the scope of the thesis. The problem that was in focus was the automation of the order fulfilment process. The careful analysis of the supply chain showed that optimizing/automating this part of the supply chain will contribute the most to reducing SC's lead time and manual labour required.

A prototype of a script for automatic order processing from our web store was developed. It is not yet fully functional, but it has the potential to significantly reduce order lead time and manual labour required to support day-to-day supply chain operations. Also, a prototype of a Zoho Creator GUI application was developed to enable packers in a warehouse to quickly enter information about a package that was shipped, which is then forwarded to Zoho's information system.

Since we didn't find any existing papers describing someone attempting something similar in Zoho's platform, we took quite a unique approach to tackle the problem of automating our supply chain. The logic behind the order processing is tailored to our particular use case, which we achieved by exploiting Zoho's adaptable low-code software development environment.

Another potential improvement could involve a cost-benefit analysis comparing the current and desired future states of the supply chain. This would help evaluate whether the expected efficiency gains, cost reductions, and performance improvements justify the investments and changes required for transformation.

The thesis provides a robust way of analysing a company's supply chain and finding potential improvements. This could help anyone attempting to do something similar in their own company. The thesis could also be used as a reference by any company using the Zoho platform to automate its order processing or tackle a similar automation problem.

5.2 Research limitations and recommendation for further research

A limitation of the current order processing "Custom function" is that handling another (third) warehouse would be very difficult. The issue would occur in the main part of the code, which iterates line items in the sales order and selects the appropriate warehouse to ship from for each line item based on the remaining stock in the primary and secondary warehouses. If we wanted to extend the code to support a third warehouse in a currently implemented way, there would need to be a lot of repeated code, and the current way of doing it just wouldn't be appropriate. The proper way of tackling the problem of supporting an arbitrary number of warehouses would be to create a recursive function. We didn't do it since it would require a lot of development time, and the complexity and amount of repetition

of the currently made code, which supports only two warehouses, is not that high. However, if someone wanted to support many warehouses, this would be one of the areas for further improvement.

The developed GUI application for packers in the warehouse also has a potential limitation. The problem is that the warehouse might use its information system and wouldn't want to force workers to use our application. In such a case, a more appropriate solution would be to connect to their WMS if they have an open API. This could be achieved with Deluge, but the downside would be that custom code would need to be developed to interface with each warehouse's WMS API.

Finally, the prototype of our order fulfilment process automation software will need to be developed into a fully functional product. Hopefully, the code will reduce the human labour required to support the order fulfilment process to almost zero and significantly reduce order lead time, leading to cheap and fast order handling in Ubiquity Robotics.

6 CONCLUSION

The examination of the company's supply chain highlighted areas for improvement. The suggestion to incorporate an additional warehouse led to better logistics management, faster delivery times, and cheaper shipping costs. The supply chain problem we focused on in this thesis was enhancing the order fulfilment process using a low-code platform - Zoho. A systematic analysis of the company's current operations made it clear that automating various steps in this process could significantly increase supply chain efficiency.

With its user-friendly interface and adaptable features, Zoho offers a practical solution for businesses looking to simplify their order fulfilment. By leveraging its tools, the company can ensure a smoother flow of information and faster response to order requests.

It is also worth noting the potential for integrating API calls within the Zoho environment. This capability shows the platform's seamless connectivity, connecting various business functions for more cohesive operations.

To wrap up, this thesis showcases the methodological approach of identifying inefficiencies in a company's supply chain and implementing solutions to these inefficiencies. The core focus was addressing one major issue: the lack of automation of the order fulfilment process. The research demonstrates the potential such platforms can have in streamlining and enhancing complex business functions by employing a low-code solution. As companies deal with increasing demand and supply chain complexities, automation becomes not just advantageous but essential for sustainable growth. This thesis offers a pathway for businesses to optimize their supply chain, especially the order fulfilment process.

REFERENCE LIST

1. Ahmad, A. N. A., Lee, T. C., Ramlan, R., Ahmad, M. F., Husin, N., & Abdul Rahim, M. (2017). Value Stream Mapping to Improve Workplace to support Lean Environment. *MATEC Web of Conferences, 135*. https://doi.org/10.1051/matecconf/201713500032
2. Anderson, J. S., Morgan, J. N., & Williams, S. K. (2011). Using Toyota's A3 Thinking for Analyzing MBA Business Cases. *Decision Sciences Journal of Innovative Education, 9*(2) 275-285. https://doi.org/10.1111/j.1540-4609.2011.00308.x
3. Ballou, R. H. (2007). The evolution and future of logistics and supply chain management. *European Business Review, 19*(4). https://doi.org/10.1108/09555340710760152
4. Bassuk, J. A., & Washington, I. M. (2013). The a3 problem solving report: a 10-step scientific method to execute performance improvements in an academic research vivarium. *PloS One, 8*(10) 1-9. https://doi.org/10.1371/journal.pone.0076833
5. Beske, P. (2012). Dynamic capabilities and sustainable supply chain management. *International Journal of Physical Distribution & Logistics Management, 42*(4), 372–387. https://doi.org/10.1108/09600031211231344
6. Boysen, N., de Koster, R., & Weidinger, F. (2019). Warehousing in the e-commerce era: A survey. In *European Journal of Operational Research, 277*(2), 396-411. https://doi.org/10.1016/j.ejor.2018.08.023
7. Britt, H. (2021, November 2). *What Is "Upstream" and "Downstream" in Supply Chain Management?* https://www.thomasnet.com/insights/what-is-upstream-downstream-supply-chain-management/
8. Croxton, K. L. (2003). The Order Fulfillment Process. *The International Journal of Logistics Management, 14*(1) 19-32. https://doi.org/10.1108/09574090310806512
9. Cui, J., Xu, J. M., Lin, H., Li, W., & Sun, Z. M. (2011). A study of rapid business application development in the cloud. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 6769 LNCS*(PART 1) (pp. 398–407). https://doi.org/10.1007/978-3-642-21675-6_46
10. Dingsøyr, T., Nerur, S., Balijepally, V., & Moe, N. B. (2012). A decade of agile methodologies: Towards explaining agile software development. *In Journal of Systems and Software 85*(6) 1213-1221. https://doi.org/10.1016/j.jss.2012.02.033
11. Duranik, T., Stopper, M., & Ruzbarsky, J. (2011). Applying value stream mapping to identify hidden reserves and avoid bottlenecks. *Annals of DAAAM and Proceedings of the International DAAAM Symposium 22*(1) 969-970. https://doi.org/10.2507/22nd.daaam.proceedings.473
12. Fernando, J. (2022, January 29). *Supply Chain Management (SCM)*. Investopedia. https://www.investopedia.com/terms/s/scm.asp
13. Heydari, M., Lai, K. K., & Zhou, X. (2020). Creating Sustainable Order Fulfillment Processes through Managing the Risk: Evidence from the Disposable Products Industry. *Sustainability, 12*(7), 2871. https://doi.org/10.3390/su12072871

14. Hiller, T., Demke, T. M., & Nyhuis, P. (2024). Throughput Time Predictions Along the Order Fulfilment Process. *IEEE Access, 12* 9705-9718. https://doi.org/10.1109/ACCESS.2024.3353029
15. Hirzel, M. (2022). *Low-Code Programming Models*. Communications of the ACM, *66*(10) 76-85 https://doi.org/10.1145/3587691
16. Lambert, D. M., & Cooper, M. C. (2000). Issues in Supply Chain Management. *Industrial Marketing Management*, *29*(1), 65–83. https://doi.org/10.1016/S0019-8501(99)00113-3
17. Lancioni, R. A., Smith, M. F., & Oliva, T. A. (2000). The role of the internet in supply chain management. *Industrial Marketing Management*, *29*(1) 45-56. https://doi.org/10.1016/s0019-8501(99)00111-x
18. Lean Enterprise Institute. (n.d.). *What is Lean?* https://www.lean.org/explore-lean/what-is-lean/
19. Lynn, R. (n.d.). *A3 process and problem solving*. Planview. https://www.planview.com/resources/guide/business-process-improvement/a3-process-problem-solving/
20. Meixell, M. J., & Gargeya, V. B. (2005). Global supply chain design: A literature review and critique. *Transportation Research Part E: Logistics and Transportation Review, 41*(6), 531–550. https://doi.org/10.1016/J.TRE.2005.06.003
21. Melnyk, S. A., Narasimhan, R., & DeCampos, H. A. (2014). Supply chain design: Issues, challenges, frameworks and solutions. *International Journal of Production Research, 52*(7) 1887-1896. https://doi.org/10.1080/00207543.2013.787175
22. Mentzer, J. T., DeWitt, W., Keebler, J. S., Min, S., Nix, N. W., Smith, C. D., & Zacharia, Z. G. (2001). DEFINING SUPPLY CHAIN MANAGEMENT. *Journal of Business Logistics*, *22*(2), 1–25. https://doi.org/10.1002/j.2158-1592.2001.tb00001.x
23. Payne, T., & Peters, M. J. (2004). What Is the Right Supply Chain For Your Products? *The International Journal of Logistics Management*, *15*(2), 77–92. https://doi.org/10.1108/09574090410700310
24. Pazour, J. (2021). *What is a Supply Chain? and How can we Design them?* [YouTube]. https://www.youtube.com/watch?v=dFZwN1DW-Jw
25. Penfield, D. (2013). *ValueStreamMapParts.png*. Wikipedia. https://commons.wikimedia.org/w/index.php?curid=28553995
26. Ramaa., A., N. Subramanya, K., & M. Rangaswamy, T. (2012). Impact of Warehouse Management System in a Supply Chain. *International Journal of Computer Applications, 54*(1), 14-20. https://doi.org/10.5120/8530-2062
27. Schrauf, S., & Berttram, P. (2016). *Industry 4.0 & How digitization makes the supply chain more efficient, agile, and customer-focused*. Strategy and PWC. https://www.strategyand.pwc.com/gx/en/insights/2016/digitization-more-efficient.html
28. Shukla, S. (2021). *What is eCommerce Order Fulfilment?* netsolutions.com. https://www.netsolutions.com/hub/ecommerce/order-fulfillment
29. Siegwart, R., Nourbakhsh, I. R., & Scaramuzza, D. (2011). *Introduction to Autonomous Mobile Robots, Second Edition.* MIT Press (Vol. 23).

30. Singh, B., Garg, S. K., Sharma, S. K., & Grewal, C. (2010). Lean implementation and its benefits to production industry. *International Journal of Lean Six Sigma*, *1*(2), 157–168. https://doi.org/10.1108/20401461011049520
31. Skjoett-Larsen, T. (1999). Supply Chain Management: A New Challenge for Researchers and Managers in Logistics. *The International Journal of Logistics Management*, *10*(2) 41-54. https://doi.org/10.1108/09574099910805987
32. Tseng, Y., Yue, W., & Taylor, M. A. P. (2005). The role of transportation in logistics chain. *Proceedings of the Eastern Asia Society for Transportation Studies*, *5*, 1657–1672. https://www.researchgate.net/publication/281230908_The_role_of_transportation_in_logistics_chain
33. Tsonev, N. (n.d.). *What Is Value Stream Mapping? Benefits and Implementation.* https://kanbanize.com/lean-management/value-waste/value-stream-mapping
34. Ubiquity Robotics (n.d.). *Ubiquity Robotics*. ubiquityrobotics.com. https://www.ubiquityrobotics.com/

APPENDICES

Appendix 1: Summary of the thesis in Slovene Language

Magistrska naloga obravnava optimizacijo dobavne verige podjetja Ubiquity Robotics, ki se ukvarja s proizvodnjo mobilnih robotov. Podjetje se zaradi hitre rasti sooča z izzivi pri organizaciji svoje dobavne verige, ki trenutno temelji na enem samem skladišču v Kitajski, od koder se izdelki pošiljajo po vsem svetu. Ta pristop povzroča visoke stroške pošiljanja, še posebej za oddaljene destinacije, saj so izdelki podjetja relativno veliki in težki, kar dodatno povečuje stroške prevoza. Stroške veča tudi metoda pošiljanja, ki je bila v času pisanja samo z letali. Poleg tega podjetje uporablja ročno obdelavo naročil, kar pomeni, da je proces dolgotrajen in nagnjen k napakam, kar še dodatno povečuje operativne stroške.

V nalogi se najprej predstavi širši pomen dobavne verige in potrebo po njeni optimizaciji, še posebej v kontekstu hitro rastočih in tehnološko naprednih industrij, kot je robotika. Dobavna veriga je namreč ključna za uspešno poslovanje, saj povezuje dobavitelje, proizvajalce in kupce ter zagotavlja, da izdelki pravočasno prispejo do končnih uporabnikov. Učinkovita dobavna veriga lahko podjetju omogoči zmanjšanje stroškov, izboljšanje storitev za stranke ter povečanje konkurenčnosti na trgu.

V okviru naloge se za analizo trenutnega stanja dobavne verige podjetja uporabi metoda analize toka vrednosti (VSM). Ta metoda omogoča vizualizacijo vseh korakov v dobavni verigi, s čimer se lažje prepoznajo ozka grla in priložnosti za izboljšave. Poleg tega se uporabi tudi A3 tehnika reševanja problemov, ki omogoča sistematičen pristop k identifikaciji težav in iskanju rešitev. Na podlagi analize je ugotovljeno, da so glavni problemi podjetja povezani z visokimi stroški pošiljanja ter neučinkovitostjo procesa obdelave naročil.

Eden izmed ključnih predlogov za izboljšanje dobavne verige je uvedba dodatnega skladišča v Kanadi, kar bi omogočilo boljšo logistično pokritost in znižanje stroškov pošiljanja, še posebaj za severnoameriške stranke. Izdelki bi se iz Kitajske v Kanado prevažali z ladjami, nato pa bi se iz kanadskega skladišča razpošiljali naprej s tovornjaki. Ta sprememba bi pripomogla k hitrejšemu in cenejšemu dostavljanju izdelkov strankam, kar bi povečalo njihovo zadovoljstvo. To spremembo je podjetje planiralo že nekaj časa, naloga pa je potrdila da se sprememba res splača.

Drugi pomemben del naloge se osredotoča na avtomatizacijo procesa obdelave naročil z uporabo Zoho-vega nizkokodnega razvojnega okolja. V okviru naloge je razvit prototip skripte za avtomatsko obdelavo naročil, ki omogoča hitrejšo in natančnejšo obdelavo naročil, s čimer se zmanjša potreba po ročnem delu in možnost napak. Prav tako je razvit uporabniški vmesnik za skladiščnike, ki omogoča enostavnejše sledenje pošiljkam in boljšo integracijo s preostalim informacijskim sistemom podjetja. Kljub temu da gre za prototip, ki še ni v celoti funkcionalen, ima sistem potencial za znatno zmanjšanje stroškov in časa obdelave naročil.

Naloga poudarja pomembnost avtomatizacije in optimizacije dobavnih verig v sodobnih podjetjih, zlasti v tistih, ki delujejo v tehnološko zahtevnih panogah. S pomočjo nizkokodnih platform, kot je Zoho-va, lahko podjetja izboljšajo učinkovitost svojih procesov, zmanjšajo stroške in povečajo konkurenčnost. Na podlagi izvedene analize in razvitih rešitev ima

podjetje Ubiquity Robotics priložnost za bistveno izboljšanje svoje dobavne verige, kar bo prispevalo k boljši poslovni uspešnosti in zadovoljstvu strank.

Appendix 2: Deluge code for processing new orders in Zoho Inventory

```
// 2. VARIABLE INITIALIZATION
// Initialization of some variables needed later
packageSkuToInfo =
{"1":{"length":"2","width":"0.6","height":"1","weight":"5"}};
warehouseIds = {"ZhenHub TsuenWan Hong Kong":"1291642000015318002","GS
Warehouse Stratford Ontario Canada":"1291642000015318158","Out of
stock":"1291642000016698035","Not selected":"1291642000016693175"};
salesorderID = salesorder.get("salesorder_id");
organizationID = organization.get("organization_id");
contactID = salesorder.get("customer_id");
lineItemsUpdated = List();
countryCode = salesorder.get("shipping_address").get("state_code");

// 3. PREFERRED WAREHOUSE SELECTION
// Initialize an empty map for query parameters
queryData = Map();

// Fetch records from Zoho Sheet with given ID and sheet name, without any
filters
response =
zoho.sheet.getRecords("el1w9165e9a2089bb417c8c5152f70ed80b0e","Sheet1",queryDat
a,"sheet_connection");

// Set default preferred warehouse
preferredWarehouseName = "ZhenHub TsuenWan Hong Kong";

// Extract shipping address and country from the sales order
shipping_address = salesorder.get("shipping_address");
country = shipping_address.get("country");

// Iterate through each record (row) from the fetched records
for each  record in response.get("records")
{
    // Check if the current row has a column that matches the country
    if(record.containValue(country))
    {
        // Iterate through each key (column header) in the record
        for each  key in record.keys()
        {
            // If the value of the column matches the country, set that column
header as the preferred warehouse
            if(record.get(key) == country)
            {
                preferredWarehouseName = key;
                break;  // Break out of the inner loop once a match is found
            }
```

```
        }
        break;  // Break out of the outer loop once a match is found
    }
}

// Based on the preferred warehouse, set the secondary warehouse
if(preferredWarehouseName == "ZhenHub TsuenWan Hong Kong")
{
    secondaryWarehouseName = "GS Warehouse Stratford Ontario Canada";
}
else if(preferredWarehouseName == "GS Warehouse Stratford Ontario Canada")
{
    secondaryWarehouseName = "ZhenHub TsuenWan Hong Kong";
}
else  // If the preferred warehouse is neither of the known ones
{
    info "ERROR, WH not recognized";  // Log an error message
}

// 4. PACKED ITEMS RETRIVAL AND UPDATE
// Fetch all packages associated with the sales order from Zoho Inventory API
packages = invokeurl
[
    url :"https://inventory.zoho.com/api/v1/packages?organization_id=" +
organizationID + "&salesorder_number_startswith=" +
salesorder.get("salesorder_number")
    type :GET
    connection:"zom"
];

// Create a map to store quantities of items that have been packed
sentItems = Map();

// Create a list to store IDs of line items that have already been packed
sentItemIds = List();

// Iterate through each fetched package
for each  package in packages.get("packages")
{
    // Fetch details of the individual package from Zoho Inventory API
    pack = invokeurl
    [
        url :"https://inventory.zoho.com/api/v1/packages/" +
package.get("package_id") + "?organization_id=" + organizationID
        type :GET
        connection:"zom"
    ];
    pack = pack.get("package");
```

```
    // Iterate through each line item in the fetched package
    for each  lineItem in pack.get("line_items")
    {
        // Add the line item's ID to the list of sent item IDs
        sentItemIds.add(lineItem.get("so_line_item_id"));

        // Extract the item's ID and warehouse ID from the line item
        item_id = lineItem.get("item_id");
        wh_id = lineItem.get("warehouse_id");

        // If this item has already been added to our sent items map...
        if(sentItems.containKey(item_id))
        {
            // ...fetch the existing map of warehouses and quantities for this
item
            item = sentItems.get(item_id);

            // If this warehouse is already in our map for this item...
            if(item.containKey(wh_id))
            {
                // ...update the quantity for this warehouse
                item.put(wh_id,item.get(wh_id) + lineItem.get("quantity"));
            }
            else
            {
                // ...otherwise, add a new entry for this warehouse with the
current quantity
                item.put(wh_id,lineItem.get("quantity"));
            }

            // Update the map of warehouses and quantities for this item in the
main sent items map
            sentItems.put(item_id,item);
        }
        else
        {
            // If this item hasn't been added to our sent items map yet...

            // ...create a new map for warehouses and quantities
            newItem = Map();

            // Add an entry for the current warehouse with the current quantity
            newItem.put(wh_id,lineItem.get("quantity"));

            // Add this new map to our main sent items map
            sentItems.put(item_id,newItem);
        }
```

```
    }
}
// 5. UPDATE UNPACKED ITEMS
// Initialize a list to store updated line items for the sales order
lineItemsUpdated1 = List();

// Loop through all line items in the sales order
for each lineItem in salesorder.get("line_items")
{
    item_id = lineItem.get("item_id");
    // The updated line item
    lineItemUpdated = {"line_item_id":lineItem.get("line_item_id")};

    // Check if the item has been packed and if the warehouse matches
    if(sentItems.containKey(item_id) &&
sentItems.get(item_id).containKey(lineItem.get("warehouse_id")) &&
sentItemIds.contains(lineItem.get("line_item_id")))
    {
        // If the item was already packed

        wh_id = lineItem.get("warehouse_id");

        // Determine the remaining quantity of the item that's packed
        sentItemsForCurrId = sentItems.get(item_id);
        num_packed_remaining = sentItemsForCurrId.get(wh_id);
        qty = lineItem.get("quantity");

        // Update the packed quantity based on how many are left to pack
        if(num_packed_remaining > qty)
        {
            // If there are more items packed than required, we reduce the
number of items that are packed.
            num_packed_remaining -= qty;
            // Update the packed remaining for the current warehouse in the
sentItems map.
            sentItemsForCurrId.put(wh_id, num_packed_remaining);
            // Update the global sentItems map with the modified packed count
for this item.
            sentItems.put(item_id, sentItemsForCurrId);
        }
        else if(num_packed_remaining == qty)
        {
            // If the number of items packed is exactly equal to the required
quantity,
            // we remove the warehouse entry as all items for this warehouse
are accounted for.
            sentItemsForCurrId.remove(wh_id);
        }
```

```
        else
        {
            // If there are fewer items packed than required, we update the
line item's quantity
            // to be the number of items that were packed.
            lineItemUpdated.put("quantity", num_packed_remaining);
            // Remove the warehouse entry since the packed items for this
warehouse have been accounted for.
            sentItemsForCurrId.remove(wh_id);

            // Prepare a new line item for the remaining unpacked items.
            lineItemUpdated2 = Map();
            lineItemUpdated2.put("item_id", item_id);
            // The quantity for this new line item is the difference between
the required and the packed items.
            lineItemUpdated2.put("quantity", qty - num_packed_remaining);
            // Set the warehouse for these unpacked items as "Not selected".
            lineItemUpdated2.put("warehouse_id", warehouseIds.get("Not
selected"));
            // Add the new line item to the list of updated line items.
            lineItemsUpdated1.add(lineItemUpdated2);
        }
    }
    else
    {
        // For items not yet packed, set warehouse as "Not selected"
        lineItemUpdated.put("warehouse_id", warehouseIds.get("Not selected"));
    }

    // Add the updated line item to the list
    lineItemsUpdated1.add(lineItemUpdated);
}

// Prepare parameters to send for updating the sales order
params = {"line_items":lineItemsUpdated1};

// Send a request to update the sales order in Zoho Inventory
ret = invokeurl
[
    url :"https://inventory.zoho.com/api/v1/salesorders/" + salesorderID +
"?organization_id=" + organizationID
    type :PUT
    parameters:params.toString()
    connection:"zom"
];
// 6. SALES ORDER STOCK UPDATE
// Fetch the sales order for up-to-date stock and line item details
so1 = invokeurl
```

```
[
    url :"https://inventory.zoho.com/api/v1/salesorders/" + salesorderID +
"?organization_id=" + organizationID
    type :GET
    connection:"zom"
];
lineItemsNew1 = so1.get("salesorder").get("line_items");

// Map to keep track of available stock for each item across all warehouses
stock = Map();
for each  lineItem in lineItemsNew1
{
    item_id = lineItem.get("item_id");
    // Check if stock details for this item were already fetched
    if(!stock.containsKey(item_id))
    {
        // Fetch stock details for the item
        item = invokeurl
        [
            url :"https://inventory.zoho.com/api/v1/items/" + item_id +
"?organization_id=" + organizationID
            type :GET
            connection:"zom"
        ];
        item = item.get("item");
        stockInWhs = Map();
        for each  warehouse in item.get("warehouses")
        {
            stockInWhs.put(warehouse.get("warehouse_name"),warehouse.get("wareh
ouse_actual_available_for_sale_stock"));
        }
        stock.put(item_id,stockInWhs);
    }
}

// Iterate over each line item to decide from which warehouse the item should
be dispatched
for each  lineItem in lineItemsNew1
{
    lineItemUpdated = {"line_item_id":lineItem.get("line_item_id")};

    // If warehouse is already selected for this item, skip it
    if(lineItem.get("warehouse_id") != warehouseIds.get("Not selected"))
    {
        lineItemsUpdated.add(lineItemUpdated);
        continue;
    }
    remainingQuantityToConsider = lineItem.get("quantity");
```

```
    // Set tax details based on country
    if(countryCode == "CA")
    {
        // California tax (its handled differently than normal tax)
        lineItemUpdated.put("tax_id","1291642000000085031");
    }
    else
    {
        // Incoterm DAP tax
        lineItemUpdated.put("tax_id","1291642000016593001");
    }

    // If package details exist for this SKU, add it to line item
    if(packageSkuToInfo.containsKey(lineItem.get("sku")))
    {
        packageDetails = packageSkuToInfo.get(lineItem.get("sku"));
        lineItemUpdated.put("package_details",packageDetails);
    }

    // Logic to decide which warehouse will dispatch the item based on stock
availability
    item_id = lineItem.get("item_id");
    stockItem = stock.get(item_id);
    stockInWH = stockItem.get(preferredWarehouseName);
    if(stockInWH >= remainingQuantityToConsider)
    {
        // if we have enought stock in primary WH, send all from it
        lineItemUpdated.put("warehouse_id",warehouseIds.get(preferredWarehouseN
ame));
        stockItem.put(preferredWarehouseName,stockInWH -
remainingQuantityToConsider);
        stock.put(item_id,stockItem);
    }
    else if(stockInWH > 0)
    {
        // If there is not enough stock in primary WH, but still more than zero
        lineItemUpdated.put("quantity",stockInWH);
        stockItem.put(preferredWarehouseName,0);
        stock.put(item_id,stockItem);
        lineItemUpdated.put("warehouse_id",warehouseIds.get(preferredWarehouseN
ame));
        remainingQuantityToConsider = remainingQuantityToConsider - stockInWH;
        stockInSecondaryWH = stockItem.get(secondaryWarehouseName);
        lineItemUpdated2 = Map();
        lineItemUpdated2.putAll(lineItemUpdated);
        lineItemUpdated2.put("warehouse_id",warehouseIds.get(secondaryWarehouse
Name));
```

```
        lineItemUpdated2.remove("line_item_id");
        lineItemUpdated2.put("item_id",lineItem.get("item_id"));
        // Check stock in secondary WH
        if(stockInSecondaryWH >= remainingQuantityToConsider)
        {
            // If there is enough stock in secondary WH for all remaining
quantity

            // Create second line item which willl be sent from secondary WH
            lineItemUpdated2.put("quantity",remainingQuantityToConsider);
            stockItem.put(secondaryWarehouseName,stockInSecondaryWH -
remainingQuantityToConsider);
            stock.put(item_id,stockItem);
        }
        else if(stockInSecondaryWH > 0)
        {
            // If there is not enough stock in the secondary WH, pack all of it
that's left

            lineItemUpdated2.put("quantity",stockInSecondaryWH);
            // We used all stock in secondary WH - set it to zero
            stockItem.put(secondaryWarehouseName,0);
            stock.put(item_id,stockItem);
            remainingQuantityToConsider = remainingQuantityToConsider -
stockInSecondaryWH;
            // Create a new line item with remaining stock to pack and set its
WH as "Out of stock"
            // (since there is not enough stock in both warehouses combined)
            lineItemUpdated3 = Map();
            lineItemUpdated3.putAll(lineItemUpdated);
            lineItemUpdated3.put("warehouse_id",warehouseIds.get("Out of
stock"));
            lineItemUpdated3.remove("line_item_id");
            lineItemUpdated3.put("item_id",lineItem.get("item_id"));
            lineItemUpdated3.put("quantity",remainingQuantityToConsider);
            lineItemsUpdated.add(lineItemUpdated3);
        }
        else
        {
            // None of the warehouses have any of required stock - set the line
item WH as "Out of stock"

            lineItemUpdated2.put("warehouse_id",warehouseIds.get("Out of
stock"));
        }
        lineItemsUpdated.add(lineItemUpdated2);
    }
    else
```

```
    {
        // If we don't have any stock in primary WH
        stockInSecondaryWH = stockItem.get(secondaryWarehouseName);
        if(stockInSecondaryWH >= remainingQuantityToConsider)
        {
            // If there is enough stock in secondary WH for all quantity

            lineItemUpdated.put("warehouse_id",warehouseIds.get(secondaryWareho
useName));
            stockItem.put(secondaryWarehouseName,stockInSecondaryWH -
remainingQuantityToConsider);
            stock.put(item_id,stockItem);
        }
        else if(stockInSecondaryWH > 0)
        {
            // If there is not enough stock in the secondary WH, pack all of it
that's left
            lineItemUpdated.put("warehouse_id",warehouseIds.get(secondaryWareho
useName));
            lineItemUpdated.put("quantity",stockInSecondaryWH);
            stockItem.put(secondaryWarehouseName,0);
            stock.put(item_id,stockItem);
            remainingQuantityToConsider = remainingQuantityToConsider -
stockInSecondaryWH;
            lineItemUpdated2 = Map();
            lineItemUpdated2.putAll(lineItemUpdated);
            lineItemUpdated2.put("warehouse_id",warehouseIds.get("Out of
stock"));
            lineItemUpdated2.remove("line_item_id");
            lineItemUpdated2.put("item_id",lineItem.get("item_id"));
            lineItemUpdated2.put("quantity",remainingQuantityToConsider);
            lineItemsUpdated.add(lineItemUpdated2);
        }
        else
        {
            // None of the warehouses have any of required stock - set the line
item WH as "Out of stock"

            lineItemUpdated.put("warehouse_id",warehouseIds.get("Out of
stock"));
        }
    }
    lineItemsUpdated.add(lineItemUpdated);
}

// Update the sales order in Zoho database with modified line items
params = {"line_items":lineItemsUpdated};
ret = invokeurl
```

```
[
    url :"https://inventory.zoho.com/api/v1/salesorders/" + salesorderID +
"?organization_id=" + organizationID
    type :PUT
    parameters:params.toString()
    connection:"zom"
];
// 7. PACKAGE GENERATION
// Retrieve the sales order details from Zoho Inventory again, to get updated
information
so = invokeurl
[
    url :"https://inventory.zoho.com/api/v1/salesorders/" + salesorderID +
"?organization_id=" + organizationID
    type :GET
    connection:"zom"
];

lineItemsNew = so.get("salesorder").get("line_items");

// Initialize a map to store package details for each warehouse
packageDetails = Map();
for each  whName in warehouseIds.keys()
{
    packageDetails.put(warehouseIds.get(whName),List());
}

// Process each line item and categorize them based on warehouse_ids
for each  lineItemNew in lineItemsNew
{
    // Skip the items that are already sent
    if(sentItemIds.contains(lineItemNew.get("line_item_id")))
    {
        continue;
    }
    packageLineItem = Map();
    packageLineItem.put("so_line_item_id",lineItemNew.get("line_item_id"));
    packageLineItem.put("quantity",lineItemNew.get("quantity"));
    cfs = {};

    // Extract custom fields for the item
    for each  cf in lineItemNew.get("item_custom_fields")
    {
        cfs.add({"label":cf.get("label"),"value":cf.get("value")});
    }
    packageLineItem.put("item_custom_fields",cfs);

    // Add the line item to the package details based on its warehouse
```

```
    l = packageDetails.get(lineItemNew.get("warehouse_id"));
    l.add(packageLineItem);
    packageDetails.put(lineItemNew.get("warehouse_id"),l);
}

// Iterate over warehouses to create packages
for each  whId in packageDetails.keys()
{
    // Skip the 'Out of stock' warehouse or warehouses with no items
    if(whId == warehouseIds.get("Out of stock") ||
packageDetails.get(whId).size() == 0)
    {
        continue;
    }

    lineItems = packageDetails.get(whId);
    totalWeight = 0.0;
    totalPrice = 0.0;
    numOfPieces = 0;

    // Calculate total weight, total price and number of pieces for the package
    for each  lineItem in lineItems
    {
        for each  cf in lineItem.get("item_custom_fields")
        {
            if(cf.get("label") == "Unit weight (kg)")
            {
                totalWeight = totalWeight + cf.get("value") *
lineItem.get("quantity");
            }
            else if(cf.get("label") == "Unit price")
            {
                totalPrice = totalPrice + cf.get("value") *
lineItem.get("quantity");
            }
        }
        numOfPieces = numOfPieces + lineItem.get("quantity");
        lineItem.remove("item_custom_fields");
    }

    // Construct the package details
    package = {"date":today.toString("YYY-MM-
dd"),"line_items":lineItems,"custom_fields":{{"label":"Total weight
(kg)","value":totalWeight},{"label":"Total","value":totalPrice},{"label":"Numbe
r of pieces","value":numOfPieces.toNumber()}}};

    // Post the package to Zoho Inventory
    pret = invokeurl
```

```
    [
        url :"https://inventory.zoho.com/api/v1/packages?organization_id=" +
organizationID + "&salesorder_id=" + salesorderID
        type :POST
        parameters:package.toString()
        connection:"zom"
    ];
}
```