TALLINN UNIVERSITY OF TECHNOLOGY

SCHOOL OF ENGINEERING

Department of Mechanical and Industrial Engineering

Affordable simulation for Meshtastic communicator production costs estimation

Kulusäästlik simulatsioon Meshtastic sidevahendi tootmiskulude hindamiseks

MASTER THESIS

Student: Iko-Eerik Uustalu
Student code: 221950MARM
Supervisor: Kristo Karjust, Professor

Tallinn 2024

AUTHOR'S DECLARATION

Hereby I declare, that I have written this thesis independently.
No academic degree has been applied for based on this material. All works, major viewpoints and data of the other authors used in this thesis have been referenced.

"20" May 2024

Author: (signed digitally)
/signature/

Thesis is in accordance with terms and requirements

"20" May 2024

Supervisor: (signed digitally)
/signature/

Accepted for defence

"20" May 2024

Chairman of theses defence commission: (signed digitally)
/name and signature/

Non-exclusive licence for reproduction and publication of a graduation thesis[1]

I, Iko-Eerik Uustalu,

1. grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis "Affordable simulation for Meshtastic communicator production costs estimation"

supervised by Kristo Karjust,

1.1 to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology (TalTech) until expiry of the term of copyright;

1.2 to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.

2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.

3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

"20" May 2024

[1] *The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the University's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.*

Department of Mechanical and Industrial Engineering
THESIS TASK

Student: Iko-Eerik Uustalu, 221950MARM (name, student code)
Study programme: MARM06, Industrial Engineering and Management (code and title)
Supervisor: Professor, Kristo Karjust, +372 6203260 (position, name, phone)

Thesis topic:
(in English) Affordable simulation for Meshtastic communicator production costs estimation
(in Estonian) Kulusäästlik simulatsioon Meshtastic sidevahendi tootmiskulude hindamiseks

Thesis main objectives:

1. Examine Lean and Agile methodologies, their development, and implementation in contemporary manufacturing to create an optimized factory layout simulation.
2. Design and implement a cost-effective simulation for a single worker production line manufacturing the Meshtastic communicator, aiming to refine equipment needs, add realism, and evaluate key performance metrics to boost efficiency in a micro-company setting.
3. Perform a financial assessment of the project, covering manufacturing costs, profitability, breakeven point, and initial investment to evaluate risks and project sustainability.

Thesis tasks and time schedule:

No	Task description	Deadline
1.	Theoretical Background: Research on Lean, Agile, simulation methods, and manufacturing layouts	2024.04.08
2.	Simulation Development: Selection of simulation software, design, execution, and performance analysis of the production line	2024.04.29
3.	Financial Analysis: Calculation of manufacturing costs, profit margins, breakeven analysis, and initial investment	2024.05.06

Language: English **Deadline for submission of thesis:** "20" May 2024a

Student: Iko-Eerik Uustalu (signed digitally)

Supervisor: Kristo Karjust (signed digitally)

Head of study programme: Kristo Karjust (signed digitally)

CONTENTS

1. INTRODUCTION

This thesis aims to evaluate production costs affordably and effectively for a new product within a one-employee micro-company, using the Meshtastic communicator as a case study. The study will estimate the cost of each product component, propose a lean manufacturing layout, and use simulation to calculate manufacturing costs. The project's profitability, including payback period and monthly operating costs, will be analyzed based on competitor analysis. The thesis aims to determine the financial feasibility and minimum demand necessary for production using simulation. This will enable small companies to make informed financial decisions in the preliminary stages of product development. By emphasizing simulation over physical prototyping, this thesis aims to deliver a strategic framework for small-scale manufacturing startups, particularly in the context of Estonia's economic landscape.

Estonia's economy heavily relies on a robust industrial sector [1]. This sector is crucial for the country's sustainable growth and potential for significant advancements by enhancing industry value and productivity [1]. In recent times, the sector has faced numerous challenges, ranging from the global health crisis starting in 2020, that disrupted supply chains and worker health, to the energy crisis in 2021, which exponentially raised production costs, negatively affecting global competitiveness [1]. Moreover, a security crisis in 2022 further limited access to some essential raw materials [1]. To overcome these challenges and elevate Estonia's industry on the global stage, there is a need for concerted efforts towards innovation and the adoption of new technologies, services, and business models facilitated by both corporate and governmental investments [1]. Central to these advancements would be a highly skilled, innovative, and qualified workforce committed to future-ready solutions [1]. Supporting this vision, Estonia's Industrial Policy for 2035 underscores the necessity of integrating digital solutions like automation into everyday business operations and encouraging constant innovative efforts for developing new products and enhancing operational efficiencies as pivotal for industrial revitalization and sustained economic progression [1].

Promoting innovation in micro-enterprises can help address the challenges faced by Estonia's industrial sector. By lowering the barrier to entry for innovation and enabling ideas to be tested more rapidly and efficiently with fewer resources, we can significantly boost inventive efforts. This thesis focuses on simplifying and reducing the cost and time associated with estimating manufacturing costs, thereby facilitating innovation in micro-enterprises.

According to the European Commission, a micro-enterprise or micro-company in the European Union is defined as a company employing fewer than 10 people, with an annual turnover and/or annual balance sheet total not exceeding 2 million euros [2]. We will be relying on the Estonian definition of a micro enterprise, defined as a private limited company with total assets up to 175 thousand euros, liabilities not exceeding the owners' equity, a single shareholder who also serves on the management board, and annual sales revenue of up to 50 thousand euros [3].

In recent times, the limitations of Public Safety and Disaster Recovery (PSDR) communications, which primarily use conventional internet connectivity methods such as landlines, mobile phones, and Land Mobile Radio, have become apparent [4]. Unlike these systems, operations in environmental monitoring like tracking animals, studying ecosystems, or managing agricultural activities can be conducted without relying on standard internet infrastructure [4]. These activities, moreover, offer insights into handling communications in scenarios devoid of usual communication channels, highlighting the necessity for considering power accessibility, environmental challenges, maintenance feasibilities, sensor specifications for weight and operational requirements, and budget constraints to ensure effective data transmission in the absence of traditional infrastructure [4].

The Meshtastic communicator was selected as the focus for our case study on estimating manufacturing costs for new products due to its open-source nature, robust community support, and relevance in the current global context. The widespread interest in these devices, coupled with the challenges many face in accessing 3D printing capabilities or sourcing complex electronic components, presents a unique opportunity to explore this product.

The thesis is structured as follows:

1. **Theory:** Theoretical underpinnings, including Meshtastic software, simulation techniques, Lean principles, Agile methodologies, and manufacturing types and layouts.
2. **Meshtastic communicator:** Constituent components, design process, 3D case selection, part selection criteria, and competitor analysis.
3. **Simulation:** Affordable software selection, production line design, process and assembly time calculation, simulation construction and execution, and key performance indicators.
4. **Financial viability analysis:** Payback period, profit at maximum capacity, minimum units to cover recurring costs, recurring costs, setup costs, and material costs.

In conclusion, this thesis aims to contribute strategically to Estonia's innovation-driven framework by focusing on small-scale manufacturing startups. By delivering a strategic framework that leans more on simulation than physical prototyping, it attempts to enable micro-enterprises to navigate the complexities of product development with greater ease and lower financial risk. The results of this study, combined with the policies and advancements made on the macro-industrial level, can potentially propel Estonia as a global competitor in the industrial sector.

Keywords: Meshtastic, simulation, SimPy, master thesis.

2. THEORETICAL BACKGROUND OF THESIS

This chapter begins by providing an overview of the Meshtastic project, an open-source initiative that leverages LoRa technology and mesh networking to create an affordable, secure, and resilient wireless communication system. We will explore the technical aspects of Meshtastic, including its use of ad-hoc networks and the LoRa physical layer, as well as its legal and licensing considerations.

Following this, we will delve into lean management principles, tracing their origins to the Toyota Production System and examining their modern applications, core concepts, and commonly used tools. These insights will guide us in designing an efficient factory layout for the production of the Meshtastic communicator.

Next, we will investigate various manufacturing processes and layouts, assessing their advantages and disadvantages to determine their impact on efficiency. This analysis will inform our decision-making process when selecting the most appropriate manufacturing approach for the Meshtastic communicator project.

Furthermore, we will explore simulation techniques, which are essential tools for predicting and refining factory layout outcomes. By applying these techniques to the Meshtastic communicator case study, we aim to optimize the production process and minimize potential issues before implementation.

Finally, we will discuss how agile methodologies, borrowed from software development, can influence lean practices in a manufacturing context. Integrating agile principles into the simulation process could lead to increased flexibility and adaptability.

Let us begin by exploring the Meshtastic project in more detail.

2.1 Overview of Meshtastic software

Meshtastic is an affordable, open-source, and encrypted wireless communication system that utilizes mesh networking and LoRa (Long Range) technology to enable long-range, low-power connectivity [4]. Designed to create a resilient communication network ideal for disaster situations and remote areas where traditional networks may fail, Meshtastic leverages inexpensive GPS radios to function as secure and long-lasting

network communicators without relying on conventional communication towers or the internet [4]. Users within the Meshtastic network can easily see the location and distance of others in the network while also having the ability to send text messages at any time [4].

The Meshtastic protocol is specifically designed for mesh networks, focusing on LoRa-based wireless communications [5]. Inspired by the RadioHead mesh routing algorithm, it emphasizes simple design and power efficiency, which are critical for battery-operated devices in remote areas [5]. This protocol allows the creation of self-organizing ad-hoc mesh networks by enabling devices to send, receive, and relay messages, thus extending the network's reach without centralized infrastructure [5]. Its effectiveness in routing and power management makes it ideal for applications in environmental monitoring, asset tracking, and remote sensor networks [5].

Built on LoRa technology, which operates on sub 1 GHz bands, Meshtastic enables communication without the need for an amateur radio license and supports broadcasts up to 1 W [6]. In Europe, Meshtastic operates on two frequency bands: 433 MHz and 868 MHz [7]. The project has chosen to utilize the 868 MHz band for its higher allowed power of +27 dBm Effective Radiated Power (ERP) within the narrower band range of 869,40 to 869,65 MHz, offering a higher ERP and a 10 percent duty cycle despite being a subset of the broader 863–870 MHz Short Range Device (SRD) Band [7].

Meshtastic devices can serve as repeaters, extending the network's range by rebroadcasting messages, with a default limit of three hops per message to prevent overload [6]. Initially designed for outdoor activities like hiking or camping, Meshtastic supports text and location sharing, as well as experimental features like low bit-rate audio [6]. The system operates on the LongFast channel by default for public communication, but users can create private channels with unique encryption keys for privacy [6]. Distinctively, Meshtastic sets itself apart with AES encryption by default, enhancing privacy and security for its users—unlike amateur radio, where encryption is generally not permitted [6].

Figure 2.1 portrays Meshtastic devices that facilitate direct communication through mesh networks using LoRa technology independently of smartphones or laptops, while also offering enhanced versatility by supporting connectivity to smartphones and laptops via Bluetooth, Wi-Fi, or USB [8].

The open-source development community has contributed significantly to Meshtastic by developing firmware and Python APIs, which ease the integration of these devices with various applications [4]. Communication between a Meshtastic device and its app is facilitated through Google Protobuf, ensuring a smooth data exchange process [4]. This

open-source project has made significant strides in providing a reliable communication method in areas where traditional networks fail, demonstrating the power of community-driven technology development [4].

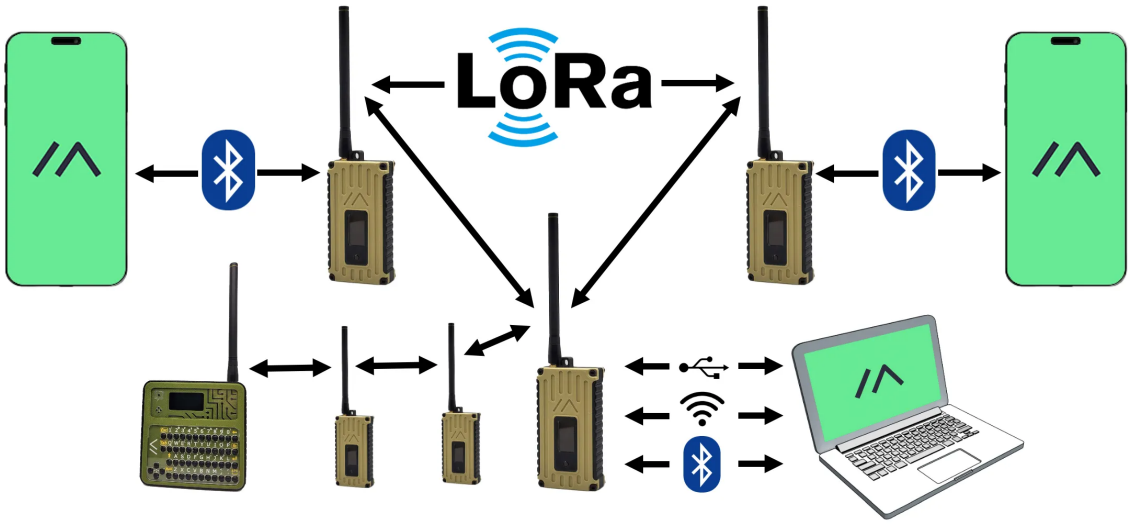

Figure 2.1 Meshtastic device connection options [8]

Ad-hoc

Meshtastic leverages the principles of ad-hoc networking to create a flexible and adaptable communication network [4]. Ad-hoc networks are a type of network where each node or hub can set itself up automatically without needing preset configurations, creating a network that is flexible and can change as needed [4]. These networks allow direct communication between all nodes, and if nodes are not directly connected, they can still communicate through other nodes via multi-hop communication [4]. The benefits of ad-hoc networks include their adaptability, cost-effectiveness, and durability [4]. However, creating a wireless mesh network, which is a form of an ad-hoc network, can be complex and faces several issues [4]. These include signal interference, limited range, security risks, along with challenges related to movement and energy use [4]. To deal with these challenges, the most commonly used protocols in these networks focus on ensuring quality of service, minimizing energy use, and being able to support growth efficiently [4].

LoRa

Meshtastic utilizes LoRa technology to enable long-range, low-power wireless communication without relying on traditional infrastructure [4]. LoRa technology stands out

as a superior option for wireless communication, surpassing alternatives like WiFi and other Low Power Wide Area Networks (LPWANs) in long-range and low-power capabilities [4]. It operates on a principle called Chirp Spread Spectrum (CSS) modulation, enabling communication over vast distances with minimal power usage [4]. Remarkably, a single LoRa base station can cover hundreds of square kilometers [4]. LoRa also incorporates a bidirectional communication protocol that works on its specific physical layer, ensuring reliable message delivery by managing connections and scheduling between endpoints and gateways [4]. LoRa's effective communication range stretches from 2 to 5 kilometers between end nodes and can extend from 10 to 15 kilometers in Long Range Wide Area Network (LoRaWAN) setups, accommodating up to 1000 nodes [4]. This technology operates on freely available industrial, scientific, and medical (ISM) radio bands, which range from 433 MHz to 923 MHz, varying by country, illustrating its potential for off-grid, decentralized communication systems globally [4].

2.1.1 Overview of legal and licensing

Meshtastic legally disclaims any warranty, stating that the software is provided "as is" and should be used at your own risk [9], [10]. The devices associated with Meshtastic are not produced by the project team and lack formal testing by regulatory bodies like UL or the FCC, rendering their use experimental and subject to caution [9], [10].

Meshtastic does not have plans for commercialization but supports the use of its software for commercial purposes under the GNU General Public License (GPLv3), emphasizing openness and collaboration [9], [10]. Meshtastic® is a registered trademark of Meshtastic LLC, and its software components are available under various licenses detailed on GitHub, provided without warranty and under a "use at your own risk" caveat [9], [10]. Users are granted a non-exclusive license to use the Meshtastic logo (Figure 2.2) and trademark, subject to certain conditions and guidelines [9], [10].

Figure 2.2 Meshtastic logo, which is a registered trademark of Meshtastic LLC [11]

The GPLv3, under which Meshtastic software is licensed, ensures that software remains free for all users, encompassing the freedoms to use, modify, share, and distribute the software [12]. By using GPLv3, the Meshtastic project commits to a philosophy that prioritizes user freedom [12].

2.2 Overview of simulation

Simulation is the process of mimicking a real-world system's operation through generating an artificial history to study its characteristics [13, pp. 1-2]. By developing a simulation model based on assumptions and relationships within the system, researchers can explore hypothetical scenarios and predict the impact of changes on system performance [13, pp. 1-2]. This method is valuable for both analyzing existing systems and designing new ones [13, pp. 1-2]. While some simple models can be solved with mathematical techniques, complex systems often require computer-based simulations to estimate their performance metrics [13, pp. 1-2].

Simulation has become a highly utilized tool in operations research due to the availability of special-purpose languages, increased computing power at lower costs, and advances in methodologies [13, p. 2]. It is particularly useful for analyzing complex systems, experimenting with changes, and enhancing understanding of system dynamics [13, p. 2]. Simulation allows for the testing of new designs, policies, and the verification of analytic solutions without the real-world costs or disruptions [13, p. 2]. It serves as a valuable educational tool, provides insights into variable importance, and aids in visualizing system operations through animation [13, p. 2].

Simulation is appealing because it replicates real or potential systems, allowing for the direct comparison of simulated outputs with real-world data [13, pp. 3-4]. Unlike optimization models, simulations are run with various inputs to observe outcomes, making them a preferred method for problem solving [13, pp. 3-4]. Advantages include testing new procedures or designs without disruption or investment, exploring hypotheses, and understanding system dynamics [13, pp. 3-4]. However, challenges include the need for specialized training, difficulty in interpreting results, time, and expense [13, pp. 3-4]. Despite these drawbacks, advancements in simulation software and hardware have made simulations more accessible and efficient, often being the only viable option for analyzing complex systems [13, pp. 3-4].

Simulation is not always the most appropriate tool [13, pp. 2-3]. Simulation should be avoided when the problem can be easily solved with common sense or analytically, when direct experimentation is cheaper, when the cost of simulation exceeds its benefits, when resources or time are insufficient, when there's a lack of data, inability to validate the model, unrealistic expectations from management, or when the system behavior is too complex or undefined to model effectively [13, pp. 2-3].

History of simulation

The development of simulation technology has been closely tied to the advancements in computing over the past several decades [14]. The history of simulation technology from the 1950s to the 1980s can be divided into four phases: pioneering, innovation, revolution, and evolution [14]. Initially, simulation technology evolved alongside early computing advancements, with basic machine code simulations emerging in the 1950s [14]. The 1960s introduced programming languages and specialized software like GPSS, enhancing simulation capabilities [14]. The 1970s saw further innovations with new programming languages and the introduction of microcomputers, laying the groundwork for the significant advancements in the 1980s [14]. This period marked a revolution in simulation technology, driven by the widespread availability of microcomputers and visual interactive simulation (VIS) software, making simulation tools more accessible to organizations [14]. The evolution continued into the 1990s and beyond, with personal computers becoming more powerful and affordable, and simulation technology benefiting from Windows technology and, to a lesser extent, the worldwide web [14]. This era saw the development of visual interactive modeling systems (VIMS), optimization techniques, virtual reality, software integration, and expanded use in the service sector [14].

2.2.1 Types of simulation

When constructing a simulation model, it's crucial to determine its key characteristics based on the system being modeled [15]. This involves choosing between different foundational types of simulations [15].

The foundational types of simulations include:

- **Stochastic vs Deterministic Simulations:** Deterministic simulations yield predictable outcomes from given inputs, while stochastic simulations incorporate randomness in inputs and outcomes [15].

- **Static vs Dynamic Simulations:** Static simulations represent a system at a single point in time, whereas dynamic simulations model the system's evolution over time [15].
- **Discrete vs Continuous Simulations:** Dynamic simulations are further divided into discrete, where changes occur at specific times, and continuous, where changes occur continuously over time [15].

Agent-based, discrete event, and system dynamics simulations are specialized simulation methodologies that build upon the foundational types of simulations, with agent-based models often incorporating both stochastic and dynamic elements to simulate the interactions of individual agents, discrete event simulations focusing on the evolution of systems at specific points in time, and system dynamics providing a framework for continuous simulation of complex systems over time, blending elements of both continuous and dynamic simulations to understand and predict system behaviors.

Agent based simulation

Agent-based modeling (ABM) involves using autonomous agents with defined behaviors to interact within an environment and with each other [16, pp. 52-54]. These agents, which can represent a wide range of entities such as people, vehicles, or organizations, operate based on rules that allow for reactive or adaptive actions [16, pp. 52-54]. ABM is useful for studying complex systems with nonlinear interactions, leading to emergent behaviors that arise from the collective actions of agents [16, pp. 52-54]. Although ABM is relatively new, especially in engineering, it has found applications in various fields [16, pp. 52-54]. These include socioeconomic systems, urban planning, healthcare, disaster response, and transportation [16, pp. 52-54]. It offers a unique approach to understanding and optimizing complex systems and has been utilized in engineering to explore system architectures and behaviors [16, pp. 52-54].

Agent-based modeling uses object-oriented programming to encapsulate data and methods within objects, allowing them to interact and manipulate their own data [16, p. 54]. Agents' behaviors and interactions can be described by equations or decision rules like if-then statements [16, p. 54]. The typical elements of agent-based models are shown in Table 2.1, but may vary across different tools and implementations [16, p. 54].

Table 2.1 Elements of agent based simulation model [16, Tab. 3.3]

Element	Description
Population	An interacting group or collection of agents during a simulation described by the type of agents and the population size.
Agent	An individual actor in a population governed by its own rules of behavior.
State	An on/off switch for an agent. When the state is active, the agent is in that state and vice versa. One or more sets of states may be placed in an agent.
Transition	Transitions move agents between states in a model. When a transition is activated, an agent moves from one state to another. They are configured by what triggers the transition. Some different types of triggers may include a timeout, a probability, or a condition based on logical relationships to other agents or events.
Action	An action manipulates agents during a simulation. They are defined by what triggers the action and what the action does when triggered. An action can be to move an agent, to change a primitive's value, to add a connection to an agent, or other.

System dynamics simulation

Continuous system models utilize differential or algebraic equations to simulate the state of a system over time, showing how variables change smoothly across small, equal time intervals [16, pp. 36-38]. These models can also be used for discrete systems by averaging variable values [16, pp. 36-38]. Euler's method and the Runge-Kutta method are common numerical techniques for integrating these equations over time to predict future states [16, pp. 36-38]. The Runge-Kutta method provides greater accuracy by using additional gradients [16, pp. 36-38]. System dynamics, a widely used continuous simulation approach developed by Jay Forrester, focuses on modeling complex systems to enhance management decisions [16, pp. 36-38]. It operates on principles of feedback loops and circular causality, using levels (accumulations) and rates (flows) to describe systems dynamically and improve processes through structural changes [16, pp. 36-38]. The continuous view abstracts individual entities into aggregated flows and does not track discrete events, emphasizing the overall behavior patterns rather than specifics [16, pp. 36-38].

Table 2.2 summarizes the elements of system dynamics models and introduces their standard graphical notations [16, pp. 38-39]. These elements can be combined to create larger infrastructures with specific behaviors, saving time and effort in model development [16, pp. 38-39]. Infrastructures can also be easily modified for various modeling purposes [16, pp. 38-39].

Table 2.2 Elements of system dynamics simulation model [16, Tab. 3.1]

Element	Description
Level	A level is an accumulation over time, also called a stock or state variable. It can serve as a storage device for material, energy, or information. Contents move through levels via inflow and outflow rates. Levels represent the state variables in a system and are a function of past accumulation of rates.
Source/Sink	Sources and sinks indicate that flows come from or go to somewhere external to the system or process. Their presence signifies that real-world accumulations occur outside the boundary of the modeled system. They represent infinite supplies or repositories that are not specified in the model.
Rate	Rates are also called flows; the actions in a system. They effect the changes in levels. Rates may represent decisions or policy statements. Rates are computed as a function of levels, constants, and auxiliaries.
Auxiliary	Auxiliaries are converters of input to output, and help elaborate the detail of stock and flow structures. An auxiliary variable must lie in an information link that connects a level to a rate. Auxiliaries often represent score-keeping variables.
Information Link	Information linkages are used to represent information flow (as opposed to material flow). Rates, as control mechanisms, often require connectors from other variables (usually levels or auxiliaries) for decision making. Links can represent closed-path feedback loops between elements.

Discrete event simulation

Discrete event simulation models systems as sequences of events that change the state of individual entities over time [16, pp. 38-48]. These entities, with unique attributes, navigate a system of interconnected nodes, utilizing resources and forming queues [16, pp. 38-48]. Instantaneous changes in entity states with simulated time reflect real-world phenomena, emphasizing the flow through processes and system interactions [16, pp. 38-48]. Utilizing tools like graphical modeling and programming languages, this approach offers functionalities for entity creation, movement, resource usage, and accumulation [16, pp. 38-48]. Key aspects, including entity paths, resource use, and queue accumulation, are essential for understanding system behavior and choosing appropriate modeling tools [16, pp. 38-48]. Standard discrete event simulation elements are shown in Table 2.3 [16, pp. 38-48].

In discrete systems, entity dynamism is highlighted by their movement and interaction, affecting the system's state over time [16, pp. 38-48]. Entities are created at specified intervals and follow paths that may lead to delays or queue waiting [16, pp. 38-48].

Priority schemes manage resource allocation in these situations [16, pp. 38-48]. The system processes entities using resources like servers, leading to accumulation points managed through specific priorities [16, pp. 38-48]. Operational dynamics, including variable management, entity manipulation, and performance data collection, ensure a structured environment for simulating real-world processes [16, pp. 38-48].

Table 2.3 Elements of discrete event simulation model [16, Tab. 3.2]

Element	Description
Create node	Generator of entities to enter a system.
Terminate node	Departure point of entities leaving the system.
Activity node	Locations where entities are served and consume resources.
Entity	An object in a system whose motion may result in an event.
Resource	Commodities used by entities as they traverse in a system.
Path	Routes that entities travel between nodes.
Batch/Unbatch Nodes	Where entities are combined into groups or uncombined.
Information link	Data connections between model elements for logical and mathematical operations.

2.3 Overview of Toyota production system

The Toyota Production System (TPS) is a distinctive approach to manufacturing that has significantly influenced the lean production movement over the last three decades or more [17, p. 12]. Originating from Toyota's post-World War II challenges, TPS diverged from the mass production strategies of companies like Ford and GM, which focused on economies of scale and large volume production [17, p. 12]. Toyota was operating in a smaller postwar Japanese market [17, p. 12]. This required the ability to produce a variety of vehicles on the same assembly line, necessitating a high degree of flexibility [17, p. 12].

This need led to a pivotal realization: prioritizing short lead times and maintaining flexible production lines resulted in not only higher quality and better customer responsiveness but also improved productivity and equipment utilization [17, p. 12]. These insights laid the groundwork for Toyota's global success in the 21st century [17, p. 12]. The evolution of the Toyota Way and TPS emerged from decades of practical experience and learning, tailored to solve Toyota's unique challenges [17, p. 12]. The Toyota Production System (TPS) represents a philosophy that, while appearing similar to traditional industrial

engineering methods in its pursuit to eliminate waste, often adopts approaches that are seemingly counter intuitive yet essential [17, p. 13]. Here are some key aspects: [17, p. 13]

- **Halting Production to Prevent Overproduction:** TPS sometimes advocates for stopping machines and ceasing part production [17, p. 13]. This is done to avoid overproduction, which is considered the most significant form of waste in the TPS approach [17, p. 13].
- **Maintaining Inventory to Stabilize Production:** TPS often suggests keeping a stockpile of finished goods, which is contrary to producing based on the fluctuating demands of customer orders [17, p. 13]. This helps in leveling out the production schedule [17, p. 13].
- **Balancing Overhead and Direct Labor:** In TPS, there is an emphasis on providing high-quality support to value-adding workers, akin to supporting a surgeon during a critical operation [17, p. 13]. This involves having dedicated team leaders who are readily available to assist when needed, illustrating a preference for selectively adding overhead rather than relying solely on direct labor [17, p. 13].
- **Matching Production Pace with Customer Demand:** Keeping workers constantly busy or pushing them to work as fast as possible is not a priority in TPS [17, p. 13]. The focus is on aligning production rates with customer demand, as pushing for maximum output can lead to overproduction and potentially increase overall labor requirements [17, p. 13].
- **Selective Use of Automation and Technology:** TPS advocates for a balanced approach towards automation and manual processes [17, p. 13]. While automation can reduce headcount and seem cost-effective, people are considered the most adaptable resource [17, p. 13]. They have the ability to continually improve processes, something automation can’t replicate [17, p. 13].
- **Methodical Planning and Implementation:** TPS favors a thorough and slow planning process, followed by experimentation and efficient deployment [17, p. 13]. This approach often yields faster and more effective results compared to hasty decision-making and immediate implementation [17, p. 13]. Toyota is known for detailed planning and piloting new practices before widespread adoption, ensuring fast and efficient rollout when ready [17, p. 13].

2.3.1 Overview of 7 wastes

The Toyota Production System (TPS) upholds one-piece flow as its ultimate goal, emphasizing uninterrupted and rework-free progression of value from inception to customer delivery [17, p. 29]. Obstacles to this flow are considered waste [17, p. 29]. Toyota has identified seven key forms of waste that do not add value in manufacturing processes [17, p. 29]. Interestingly, these principles can be adapted beyond manufacturing, finding relevance in areas such as product and software development, hospital operations, and various office processes, with slight adjustments [17, p. 29].

The following are the seven wastes and their explanations:

1. **Overproduction** - refers to the practice of producing more goods or services than currently demanded, leading to inefficiencies like surplus staffing, and extra costs for storage and transport due to excessive inventory [17, p. 29].
2. **Waiting** - waiting involves idle time when employees or machines are not actively engaged in production [17, p. 29]. This can occur while awaiting essential inputs, machine availability, or when there are no pressing deadlines [17, p. 29].
3. **Transport** - unnecessary transport describes the inefficient movement of work-in-progress (WIP) over long distances, including the movement of materials or information into or out of storage or between different stages of a process [17, p. 29].
4. **Over-processing** - incorrect processing happens when there are superfluous steps in a process, often due to sub optimal tool or product design [17, p. 29]. This inefficiency can lead to unnecessary actions, defects, and the production of higher quality than needed [17, p. 29].
5. **Inventory** - excess inventory includes having more raw materials, work-in-progress, or finished products than necessary [17, p. 29]. This leads to increased lead times, risk of obsolescence, potential damage to goods, and elevated costs for transportation and storage [17, p. 29]. Additionally, surplus inventory can mask underlying issues like production imbalances, supplier delays, defects, equipment failures, and lengthy setup times [17, p. 29].
6. **Motion** - Unnecessary movement is any non-productive physical activity by employees, such as searching for, retrieving, or organizing parts, tools, and so on [17, p. 30].

7. **Defects** - defects encompass the production of faulty items and the subsequent need for repair, rework, or replacement [17, p. 30]. This results in wasted time, effort, handling, and resources [17, p. 30].

Several authors, including Liker, have pointed out that in process industries, there is an often overlooked element of waste: the waste of human potential and creativity [18]. This element is frequently referred to as the eighth type of waste [18].

Not making the most of people's talents and skills can negatively impact an organization [19]. On the other hand, recognizing and using these abilities can bring great benefits, including outstanding performances from employees at all levels [19]. Ignoring this aspect can be harmful to the company [19].

Ohno identified overproduction as the primary form of waste in manufacturing, which might appear surprising at first [17, p. 30]. This is because overproduction inherently leads to other types of waste [17, p. 30]. When production exceeds customer demand, it results in excess inventory accumulating downstream [17, p. 30]. This excess not only ties up resources but also promotes inefficiencies in subsequent processes [17, p. 30]. For instance, large buffer inventories between processes can diminish the urgency to enhance operational efficiency [17, p. 30]. There's less incentive to conduct preventive maintenance or address quality issues promptly, as the immediate impact on final assembly is minimal [17, p. 30]. However, this approach can backfire, as it may take weeks for the repercussions of these quality lapses to become apparent, by which time significant quantities of defective products may have been produced [17, p. 30]. Thus, overproduction, by creating a domino effect of inefficiencies, is considered the fundamental waste in manufacturing [17, p. 30].

2.4 Overview of Lean manufacturing

Lean manufacturing traces its roots to post-World War II Japan [20]. Faced with limited resources to invest in manufacturing facilities comparable to those in the United States, Japanese manufacturers, particularly Toyota, embarked on a journey to significantly refine their manufacturing processes [20]. This initiative aimed to reduce waste in all operational aspects [20]. Pioneered by Taiichi Ohno and Shigeo Shingo at Toyota, lean manufacturing, also referred to as the Toyota Production System (TPS), has since been acknowledged for providing organizations with notable cost and quality advantages over traditional mass production methods [20].

The concept of Lean originated in a 1988 Sloan Management Review article, where the Toyota Production System (TPS) was described as Lean [21]. This term was chosen to highlight the TPS's efficiency and lower resource usage compared to Western production systems [21]. Lean's popularization is credited to the 1990 book The Machine That Changed The World, which emerged from the MIT-based International Motor Vehicle Programme [21]. This book revealed a significant performance gap between Japanese and Western car manufacturers and had a substantial impact on management thinking [21].

Lean can be defined as a system that uses less of everything compared to mass production, emphasizing efficiency and resourcefulness [21]. Although the The Machine That Changed The World is seen as the starting point of the Lean movement, Lean practices were already present in the USA under different names in the early 1980s [21].

2.5 Principles and Critiques of Lean Manufacturing

There are considered to be five key Lean principles aimed at improving efficiency and reducing waste in organizations [22]. These principles are: [22]

1. Determine what your end customer values for each product family [22].
2. Map out every step in producing each product family and remove steps that do not add value [22].
3. Ensure that the steps adding value occur one right after another, allowing the product to flow smoothly towards the customer [22].
4. Let the customer's demand pull the product through each step of the process [22].
5. Keep refining the process by specifying value, identifying and eliminating waste, improving flow, and responding to customer pull, with the goal of achieving perfect value delivery with no waste [22].

Despite its popularity, Lean lacks a clear, universally accepted definition, leading to varied interpretations and applications [21]. The concept has evolved beyond manufacturing to include various organizational functions and sectors [21]. Lean is considered polymorphic, meaning its interpretation varies over time and among different individuals [21].

Lean has faced criticism for lacking a solid theoretical foundation [21]. The ”Theory of Swift, Even Flow” has been proposed to underpin Lean, emphasizing the efficiency of material flow in a process [21].

2.6 Lean tools overview

In the field of lean production, also referred to as manufacturing without waste, the primary goal is to enhance resource efficiency by minimizing waste [23]. Central to lean production is the use of various tools and techniques designed to optimize operational outcomes [23]. These can include tools such as 5S, kaizen, and cellular layout [23]. The effective selection and implementation of these tools can lead to significant improvements in operational performance, evidenced by higher quality, reduced inventories, and shorter throughput times [23]. Recent research has documented a vast array of over 55 distinct lean production tools and techniques [23]. This extensive toolbox provides industries with a range of options to improve their operational efficiency and reduce waste [23].

For the simulation to be accurate, it has to follow real rules, it has to have a defined process and there has to be order to that process. So we have to design the process. Considering the design will be from scratch, it would be prudent to design it and the factory layout with lean principles in mind. But lean manufacturing is more than just the principles, there are many best practices and tools that can be used to inform these design decisions. The following are names and descriptions of common lean tools that will be used in the design of the process and manufacturing layout.

5S

The Five Ss, originating from Japanese workplace practices, are key concepts in lean production and visual management [24, p. 21]. They consist of:

- **Seiri (Sort):** This involves distinguishing between necessary and unnecessary items in the workplace, such as tools, materials, and documents, and disposing of the latter [24, p. 21].
- **Seiton (Straighten):** It focuses on orderly arrangement of the remaining items, ensuring everything has a designated place and is kept there [24, p. 21].

- **Seiso (Shine):** This step is about maintaining cleanliness through regular cleaning and washing [24, p. 21].
- **Seiketsu (Standardize):** This principle emphasizes maintaining standards of cleanliness and orderliness by routinely applying the first three Ss [24, p. 21].
- **Shitsuke (Sustain):** This is about instilling discipline to continuously implement the first four Ss [24, p. 21].

Additionally, some practitioners of lean methodologies advocate for a sixth 'S', Safety, which involves establishing and adhering to safety protocols in both workshop and office environments [24, p. 21].

Cellular Manufacturing

An example of Cellular Manufacturing is shown in Figure 2.3.

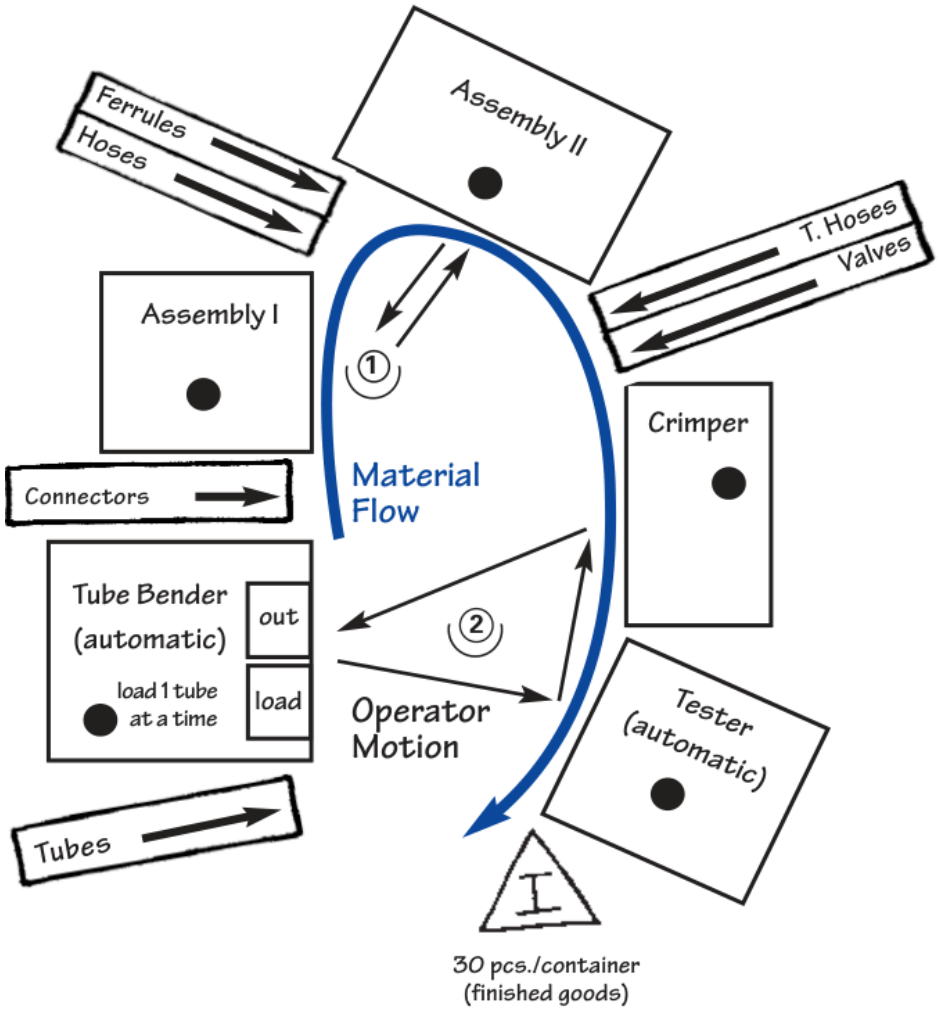

Figure 2.3 U-shaped cell [24, p. 7]

A ”Cell” in lean manufacturing refers to a setup where processing steps for a product are located immediately adjacent to each other [24, pp. 7-8]. This arrangement allows for a nearly continuous flow of parts, documents, etc., either one at a time or in small batches, through the entire sequence of processing steps [24, pp. 7-8]. A common design for such cells is a U-shape, which reduces walking distance and offers flexibility in work task combinations for operators [24, pp. 7-8]. This flexibility is crucial in lean production as it allows for adjustment of operator numbers in response to demand changes [24, pp. 7-8]. The U-shape also aids in maintaining a smooth workflow, as it enables the same operator to perform the first and last steps of the process [24, pp. 7-8].

Continuous Flow

Continuous Flow is a production method where individual items, or small and consistent batches, are produced and moved through a series of processing steps in as uninterrupted a manner as possible [24, p. 10]. Each stage in this process produces exactly what is requested by the subsequent stage [24, p. 10]. This approach can be implemented in various forms, including moving assembly lines and manual cells [24, p. 10]. It is also known by other names such as one-piece flow, single-piece flow, and make one, move one [24, p. 10]. Figure 2.4 shows a continuous one piece flow.

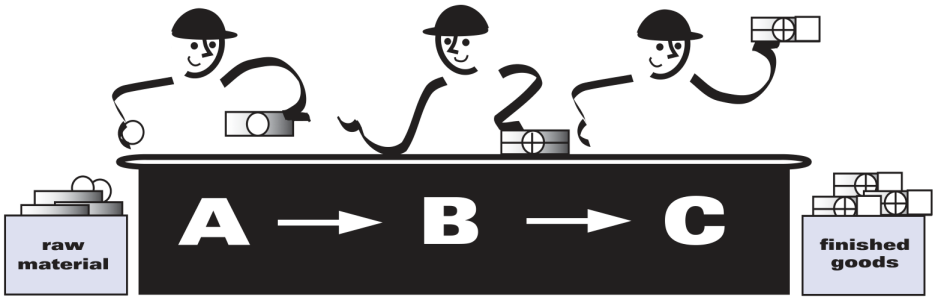

Figure 2.4 Example of continuous flow [24, p. 10]

Kanban (Pull System)

Kanban is a Japanese system used in manufacturing for signaling and instructing both production and conveyance of items in a pull system [24, pp. 42-46]. Commonly, kanbans are represented through cards, but can also include other forms like metal plates or electronic signals, all serving to avoid incorrect instructions [24, pp. 42-46]. The primary functions of kanban are to direct production processes on what and how much to produce and to guide the movement of products [24, pp. 42-46]. There are two main types: production kanban, which instruct upstream processes, and withdrawal kanban, used by operators at downstream processes [24, pp. 42-46].

In production, kanban can take various forms, with triangle kanban being a standard in lean manufacturing for batch processes [24, pp. 42-46]. These systems are critical in creating an efficient pull system, ensuring that production and material movement are based on actual demand [24, pp. 42-46].

For effective utilization of kanban, there are six key rules: (1) ordering goods in exact amounts as indicated, (2) producing goods in precise amounts and sequences, (3) making or moving items only with a kanban, (4) attaching a kanban to all parts and materials, (5) not sending defective parts or incorrect quantities to the next process, and (6) carefully reducing the number of kanban to lower inventory and highlight issues [24, pp. 42-46].

KPIs (Key Performance Indicators)

Key Performance Indicators (KPIs) are critical in business for measuring performance [25]. These benchmarks help assess the success of various organizational activities [25]. Effective KPI implementation requires selecting indicators that mirror the organization's goals [25]. KPIs are instrumental in identifying performance shifts, allowing for prompt adjustments [25]. The challenge lies in choosing relevant KPIs, as unsuitable ones can misdirect efforts [25].

2.7 Overview of different types of manufacturing processes and layouts

In this section, we'll have a look at different types of manufacturing processes and layouts. It's important to note the theory behind this for simulation, because different processes and different layouts are very suitable for different types of simulation, such as a project-based process could be difficult to simulate, but if simulated, would probably require an agent-based solution, while a continuous process, let's say like chemical manufacturing or oil refineries, are very well suited for systems dynamic simulation. Thus, getting this overview will better inform us about the type of process we are going to simulate and the layout type.

2.7.1 Manufacturing process types

The concept of manufacturing process types is determined by a process's position on the volume-variety continuum, which influences its design and management strategies [26, p. 102]. The types of manufacturing processes include Project Processes, Jobbing Processes, Batch Processes, Mass Processes, and Continuous Processes [26, p. 102]. A visual representation of how manufacturing process types align with different points on the volume-variety spectrum is provided in Figure 2.5 [26, p. 102].

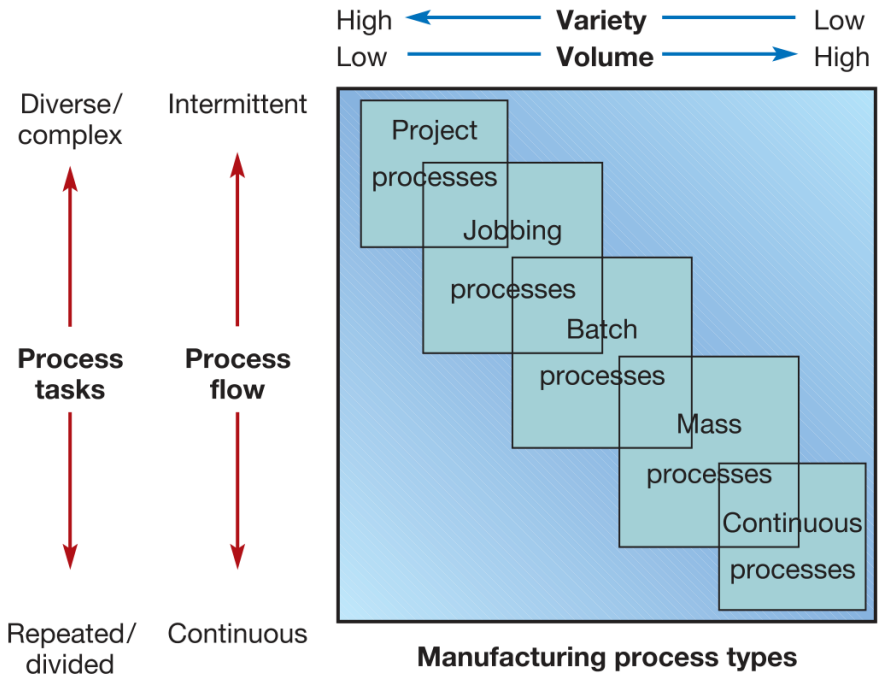

Figure 2.5 Different process types and their relation to volume-variety continuum [26, p. 102]

Project processes

Project processes are characterized by their focus on unique, often highly-customized products with a long timeline for completion [26, pp. 102-103]. Each project has a clear start and finish, typically involving low volume but high variety [26, pp. 102-103]. The tasks within these processes can be ambiguous and uncertain, requiring specific organization and transformation of resources for each individual item [26, pp. 102-103]. The complexity arises partly from the significant discretion allowed in professional judgment [26, pp. 102-103]. Examples of industries that use project processes include software design, movie production, construction, and large-scale fabrication operations like turbo generator manufacturing [26, pp. 102-103].

Jobbing processes

Jobbing processes are characterized by their handling of a diverse range of products in small quantities, where resources are allocated to multiple products simultaneously rather than to individual items [26, p. 103]. This approach often leads to products that are unique and not replicated [26, p. 103]. Despite their complexity, jobbing processes tend to produce smaller items and are generally more predictable, requiring considerable expertise [26, p. 103]. This is evident in professions like bespoke tailoring, precision engineering, furniture restoration, and small-scale printing for community events [26, p. 103].

Batch processes

Batch processes are similar to jobbing processes but have less variety [26, p. 103]. They produce multiple items at a time and become repetitive, especially with large batches or familiar products [26, p. 103]. The versatility of batch processing allows it to be applied across various production scales and types [26, p. 103]. It's commonly seen in industries such as machine tool manufacturing, the creation of certain high-end frozen foods, and in producing parts for large-scale assemblies, such as in the automotive industry [26, p. 103].

Mass processes

In manufacturing, mass processes are characterized by the high-volume production of products that, despite potential variations, share basic design elements [26, p. 104]. These processes are known for their repetitive nature and predictability [26, p. 104]. Common instances of mass processes are seen in industries like frozen food manufacturing, car assembly lines, television production, and DVD creation [26, p. 104].

Continuous processes

Continuous processes are known for their high output and limited diversity, often surpassing mass processes in volume [26, p. 104]. These processes typically run for extended periods and are sometimes truly continuous, producing products in an unbroken flow [26, p. 104]. The technology used in these processes is usually capital-heavy and not very flexible, but it guarantees a steady and predictable progression of materials [26, p. 104]. While there might be instances of product storage during these

processes, their defining trait is the seamless transition of materials from one phase to the next [26, p. 104]. This is commonly seen in industries like water treatment, petrochemical refining, electricity generation, steel production, and certain types of paper manufacturing [26, p. 104].

2.7.2 Manufacturing layout types

In operations management, a layout refers to the arrangement of resources and the allocation of tasks within a process [26, p. 193]. An effective layout ensures efficient resource flow, avoiding issues like long processing times and high costs [26, p. 193]. However, redesigning a layout can be disruptive and expensive, thus it's done infrequently and requires careful planning [26, p. 193]. There are four basic types of layouts: fixed-position, functional, cell, and product (line) layout [26, p. 193]. While there is a general correlation between layout types and process types, as illustrated in Figure 2.6, it is important to recognize that a specific process type does not necessarily dictate a single optimal layout type [26, p. 193].

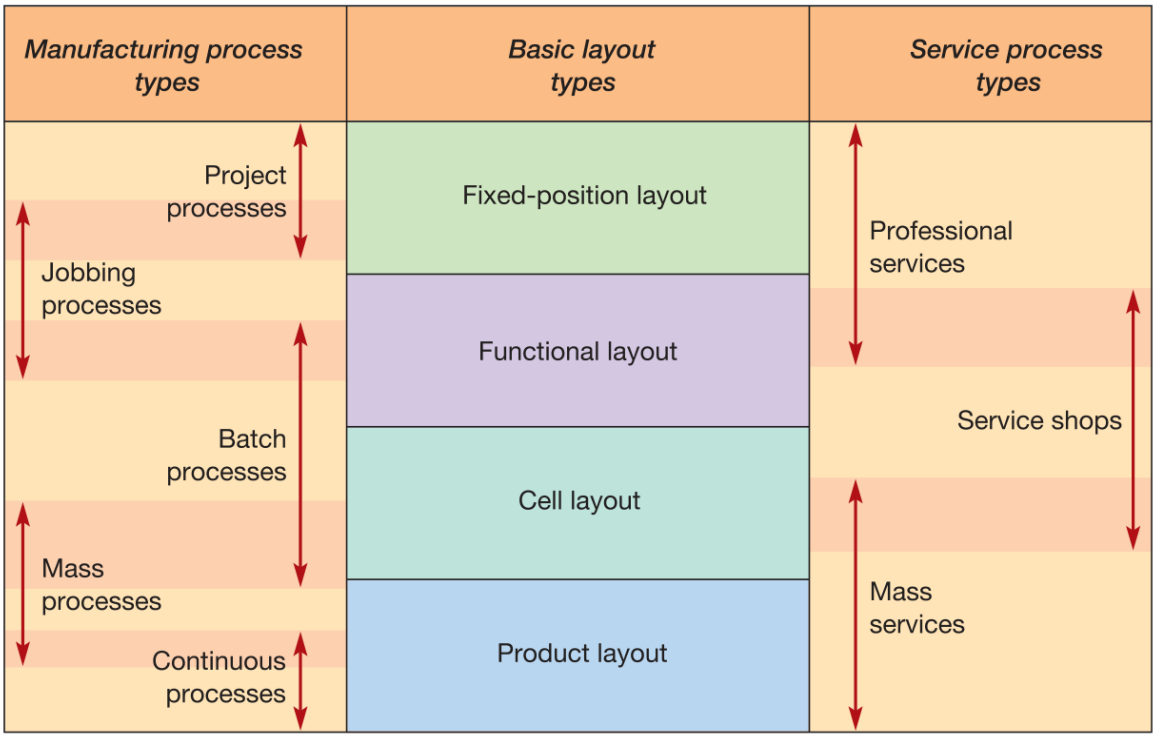

Figure 2.6 How different process types link to basic layout styles [26, p. 194]

A good layout aligns with the strategic objectives of the operation and generally aims for safety, minimized flow length, staff comfort, equipment accessibility, effective space

utilization, and long-term flexibility [26, pp. 193-201]. The choice of layout depends on the operation's volume and variety characteristics, as shown in Figure 2.7 [26, pp. 193-201]. Low volume and high variety operations, like satellite manufacturing, might use a fixed-position layout, whereas high volume and low variety operations, like assembly plants, might prefer a product-based layout [26, pp. 193-201]. Different layouts offer various advantages and are chosen based on the operation's specific needs and characteristics [26, pp. 193-201].

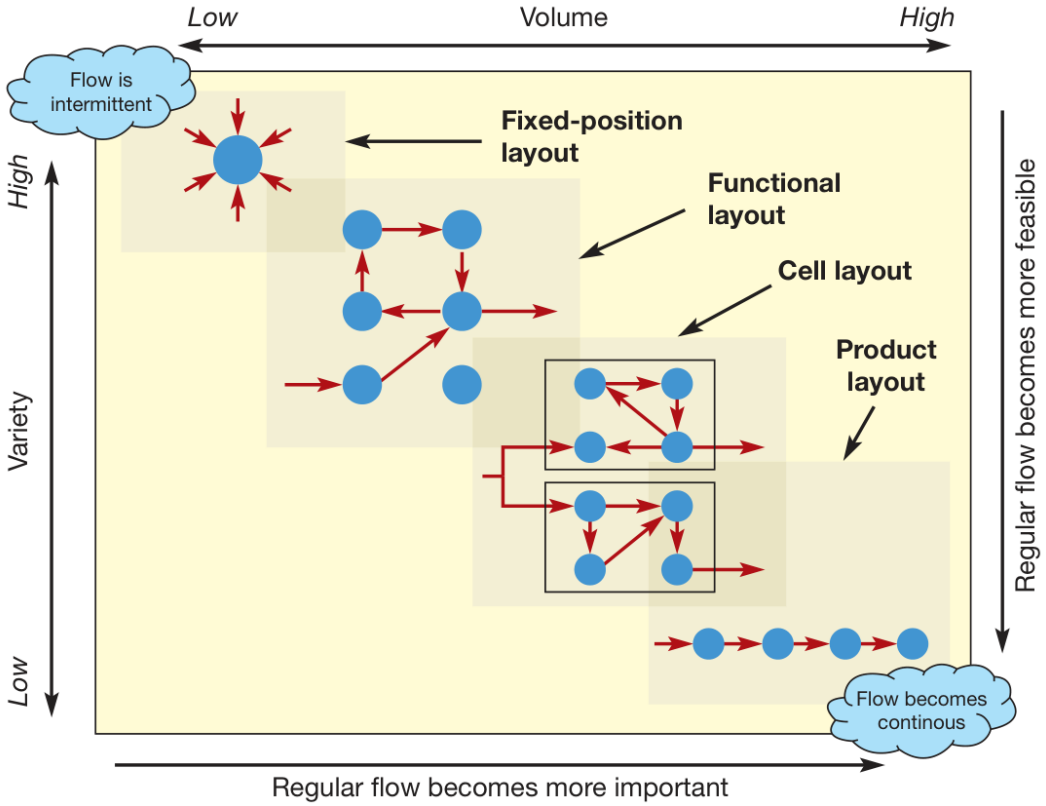

Figure 2.7 Specific process layouts are more suitable for varying model-volume combinations [26, p. 202]

Fixed-position layout

Fixed-position layout refers to a process where the object receiving the work remains stationary while the necessary equipment and personnel move around it [26, p. 194]. This method is often used when the product or service recipient is too large, delicate, or unwilling to be moved [26, p. 194]. Examples include motorway construction, open-heart surgery, high-class restaurant service, shipbuilding, and mainframe computer maintenance [26, p. 194].

Functional layout

Functional layout groups similar resources or processes together for efficiency and to improve resource utilization [26, pp. 194-196]. In such a layout, products, information, or people follow a path based on specific needs, leading to complex operation flow patterns [26, pp. 194-196]. This layout is exemplified in aircraft engine part manufacturing, where processes like heat treatment require specialized support, machining centers need technical operators, and a single grinding section handles all grinding needs [26, pp. 194-196].

Cell layout

Cell layout refers to organizing an operation into specific areas or cells, each designed to meet the immediate processing needs of selected resources or products [26, p. 197]. This setup can be designed based on function or product, and allows for resources to move systematically from one cell to another [26, p. 197]. This methodology is aimed at simplifying flow in operations with complex functional layouts [26, p. 197]. Examples of cell layouts are found in various industries, including manufacturing computer components for specific customers, arranging related grocery items for convenience in supermarkets, and centralizing services for maternity care in hospitals [26, p. 197]. The concept is applicable not only in manufacturing but also in service industries [26, p. 197].

Product (line) layout

Product layout, also known as line or flow layout, organizes transforming resources (like machinery or staff) to facilitate a seamless 'flow' or sequence that materials, information, or customers follow, mirroring their required sequence of operations [26, pp. 197-198]. This setup, prompted by standardized product or service demands, ensures clarity, predictability, and easy control [26, pp. 197-198]. Examples include automobile assembly, mass-immunization programs, and self-service cafeterias, all of which follow a common sequence of processes or customer movements [26, pp. 197-198].

Mixed layouts

Mixed layouts in operations involve the combination of various basic layout types or the utilization of 'pure' basic layouts in different sections [26, p. 198-200]. For instance,

hospitals typically use a functional layout for overall organization based on department functions (e.g., X-ray, surgery, blood-processing) but adopt different layouts within these departments, such as functional layout for X-ray, fixed-position for surgical theatres, and product layout for blood-processing labs [26, p. 198-200].

2.8 Overview of Agile methodology

The principles of Lean, which focus on eliminating waste and improving efficiency, can also be applied to software development, leading to the emergence of Agile methodology.

The Agile Software Development Manifesto, established in 2001, emphasizes a more flexible and human-centric approach to creating software [27]. It highlights the importance of prioritizing people and collaboration, producing functional software, working closely with customers, and being adaptable to change over strictly adhering to processes, extensive documentation, rigid contracts, and fixed plans [27]. Though it acknowledges some value in the latter, the manifesto clearly advocates for a stronger emphasis on the former elements to improve software development outcomes [27].

Agile Methods in software development have been in use for over two decades and focus on rapid development amidst changing requirements [28]. Key features include iterative development, frequent customer releases, close customer collaboration, and adaptability to requirement changes [28]. Agile differs from traditional methods by emphasizing small self-organizing teams, continuous design improvement, test-driven development, and continuous integration [28]. Knowledge management in agile is more tacit, relying on face-to-face communication over extensive documentation [28]. Additionally, lean manufacturing principles have significantly influenced agile development [28].

Digital wastes

In the era of Industry 4.0, it’s crucial to rethink the traditional eight types of waste from Lean manufacturing in the context of digital waste, as shown in Table 2.4 [29]. The concept revolves around the idea that just like physical waste, digital waste—stemming from excessive energy use and unchecked investments in new technologies—can drain resources [29]. The essence of Lean thinking, which focuses on eliminating waste and improving efficiency, is just as relevant in our digital operations [29]. Without applying these Lean principles, businesses embracing digital transformation may find themselves

facing high costs in areas like training, upkeep, and new tools [29].

Table 2.4 Characterization of the digital waste in the eight original Lean wastes [29]

Lean wastes	**Corresponding types of digital wastes**
Defects	Lack of precise data; Lack of precise data processing.
Over Production	Excess of data gathering; Excess of data information; Excess of technology used to maintain the data flow.
Waiting	Data don’t reach the right person at the right time
Extra Processing	Excess of energy used for generating data and managing the hardware; Over processing data that will not be used.
Motion	Lack of data transition from the company to 3rd parties’ stakeholders in logistics control.
Inventory	Excess of data storage in clouds.
Transportation	Lack of flexibility in the company communication panels.
Skills	Excess of training; Lack of feedback; Losing jobs to the machine

3. MESHTASTIC COMMUNICATOR

While the simplest Meshtastic communicators can be made with just a LoRa-compatible development board and an appropriate antenna, many versions include additional features like OLED displays, Bluetooth and LoRa antennas, GPS modules, battery packs, on-off switches, and external enclosures. OLED displays are particularly useful as they allow the device to be more independent and usable without a cell phone. The display can show the most recent message sent, the location it was sent from, the distance (if that data is available), and other information about the device's status. This feature is especially beneficial when the device is intended to be used independently of a laptop or cell phone.

3.1 Meshtastic communicator design

Our design incorporates a LoRa 868 MHz antenna for long-range communication, a Bluetooth antenna for short-range connectivity, a LoRa radio board, a battery, a power switch, and a 3D printed enclosure to house everything. The display module is included for feature parity with other available options, which usually have a display module included. The JST PH2 Power Connector Cable is used to connect the battery to the board, which can also be powered by solar energy. The 18650 Battery Holder is used to avoid the need for soldering the battery cell directly, as this can be dangerous and cause damage to the battery. This approach maintains simplicity by using only the essential components needed for off-grid communication, ensuring the device is both functional and reliable in off-grid situations. The bill of materials for the Meshtastic communicator can be seen in Table 3.1.

3.1.1 Estimating parts costs

The Bill of Materials (BOM) for the Meshtastic Communicator, presented in Table 3.1, lists all the components and materials required to assemble a single unit of the device. The BOM includes the item number, description, quantity, and cost (in euros) for each

component. The 3D printed case material costs have been added to the BOM, which are relatively affordable compared to the other components.

Table 3.1 Bill of materials cost for Meshtastic communicator

Item No.	Description	Quantity	Cost (EUR)
1	RAK4631 WisBlock core module	1	27,15 [30]
2	RAK19007 WisBlock base board	1	13,84 [31]
3	3D printed case material	1	1,33
4	LoRa antenna	1	9,87 [32]
5	18650 battery cell	1	6,93 [33]
6	18650 battery holder	1	1,44 [34]
7	LoRa antenna adapter cable	1	3,67 [35]
8	Bluetooth antenna	1	1,44 [36]
9	Display module	1	8,12 [37]
10	Miscellaneous items (heat shrink, solder, etc.)	-	2,00
11	On-off switch	1	0,69 [38]
12	JST PH2 power connector cable	1	0,70 [39]
13	M3 x 20 mm bolts	4	1,00
14	M3 nuts	4	0,50
15	M2,5 screws	4	0,50
16	M2 x 4 mm screws	4	0,50
17	M1,2 x 3 mm screws	4	0,25
		Total Cost	**79,93**

Prices will be listed without tax whenever possible. Suppliers who deliver directly to Europe, and who are more B2B orientated and who offer bulk discounts when available, were prioritized. Bulk quantities up to 100 pieces were considered in order to take advantage of bulk purchasing discounts but no more to avoid excess stock. This is important considering that the design of the communicator may be updated during production and older components would no longer be necessary.

The total cost of the components and materials required to produce a single Meshtastic Communicator unit is calculated at the bottom of the table, amounting to 79,93 euros. This cost represents the direct material expenses for each device and does not include labor, overhead, or other indirect costs associated with the manufacturing process.

The information provided in the Bill of Materials table is crucial for understanding the composition of the Meshtastic Communicator and estimating the material costs involved in its production. This data will be used to determine the overall cost of the device.

3.1.2 Processor and board

Meshtastic communicators can be built using either an nRF52 or an ESP32 processor, each with its own advantages and disadvantages [40]. nRF52-based devices are more power-efficient, making them suitable for solar-powered and portable applications [40]. On the other hand, ESP32-based devices, though less power-efficient and older, are more affordable and ideal for applications that have access to house power, need short-term portability, or require WiFi connectivity [40]. The ESP32 supports both WiFi and Bluetooth, while the nRF52 is more power-efficient, easier to update, and only supports Bluetooth [40]. Currently, the only supported development board for nRF52 is the RAK4631 Core module for RAK WisBlock modular boards, which is shown in Figure 3.1 [40].

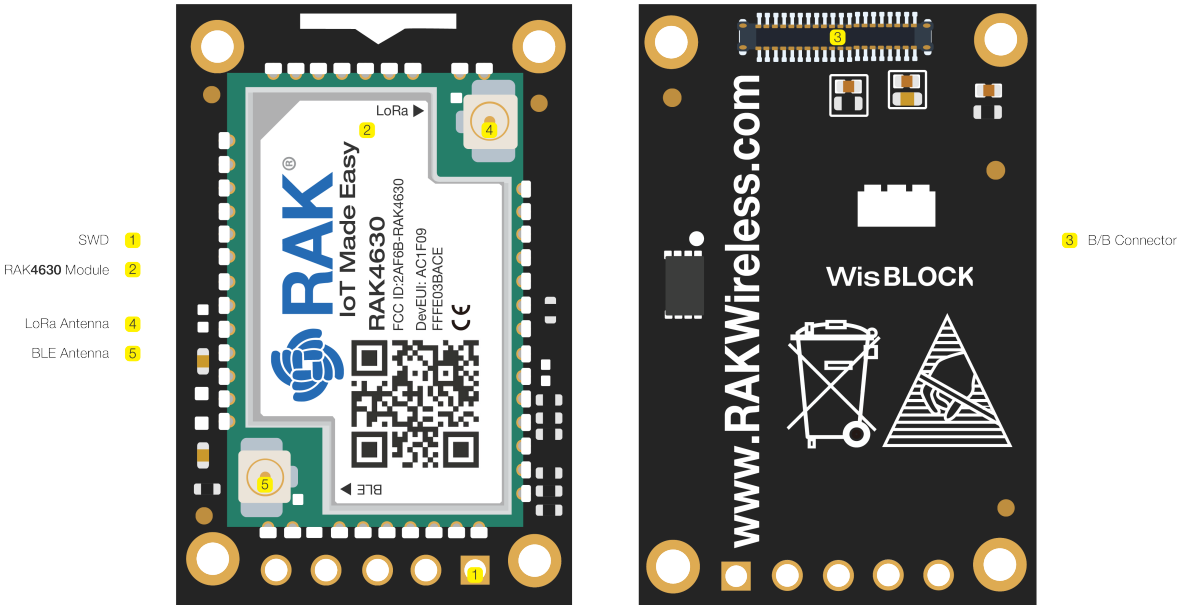

Figure 3.1 RAK4631 module overview [41]

For operation, the RAK4631 requires a WisBlock Base for power supply and programming/debug interfaces [41]. The RAK4631 module, part of the RAK WisBlock series, measures 20 mm by 30 mm [41]. It is a powerful addition with its Nordic nRF52840 MCU that enables Bluetooth 5.0 and incorporates the SX1262 LoRa transceiver by Semtech, which is more energy-efficient than its predecessors in the SX127x series [41]. Its standout features include ultra-low power usage for communications, the ease of programming through the Arduino IDE or PlatformIO for various IDEs with a 32-bit ARM Cortex-M4 CPU at 64 MHz, 1 MB Flash, and 256 KB RAM [41].

Figure 3.2 shows the RAK19007, which is a second generation WisBlock Base Board that enables connection and power supply to WisBlock Core, IO, and Modules [42]. It measures 60mm by 30mm and features a Type-C USB connector for firmware uploads,

serial communication, or battery charging, depending on the WisBlock Core used [42]. High-speed connectors ensure secure attachment and signal integrity for modules, which are also fixed with screws for reliability in vibratory environments [42].

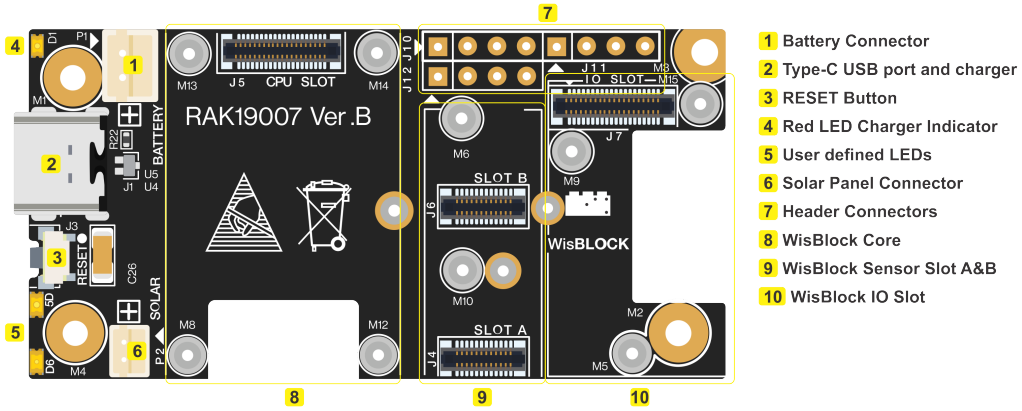

Figure 3.2 RAK19007 base board overview [42]

We have chosen the nRF52-based RAK4631 module for our design due to its power efficiency, making it suitable for solar-powered and portable applications. The RAK4631 is also easier to update and is currently the only supported development board for nRF52 that is readily available in Europe from reputable suppliers. These factors, along with the features of the RAK19007 base board, make this combination well-suited for the Meshtastic communicator design.

3.1.3 3D printed case

For estimating manufacturing costs, the decision was made to utilize an existing case design. This approach was chosen primarily due to the time-consuming nature of developing a new case for the communicator, which is not central to the current research objectives.

Additionally, adopting a pre-existing case design offers the advantage of utilizing a solution already vetted by the broader community. This not only facilitates a quicker estimation of manufacturing complexity, printing time, and assembly complexities due to its size similarity with any potential custom case but also ensures reliability in these estimations. The selected case variant Includes a screen cutout, does not include GPS functionality, and is specifically designed to accommodate 18650-type batteries.

Figure 3.3 showcases the chosen case design, which is specifically created for the RAKwireless RAK19007 and RAK5005 base boards by designer TonyG [43]. The

dimensions of the case are 103 mm in length, 50 mm in width, and 45 mm in height, ensuring a perfect fit for the specified base boards [43]. This design is protected under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License [43]. This licensing requires that credit be appropriately given to the original author, TonyG, that the work is not used for commercial purposes, and that no modified versions of the work are distributed [43].

Figure 3.3 3D printed case for RAK4631 with RAK19007 base board [43]

3.2 Meshtastic communicator competitors

The two main commercially produced Meshtastic communicator devices are the LILYGO T-Echo and B&Q Consulting Nano G1 Explorer. We are only looking at commercial devices designed to be portable.

The LILYGO TTGO T-Echo, as shown in Figure 3.4, represents an enhanced iteration of the prior T-BEAM model, specifically engineered with Meshtastic compatibility in mind [44]. This design strategically integrates advanced Bluetooth capabilities and efficient power usage, thanks to the inclusion of the nRF52840 [44]. Consumers interested in the T-Echo for Meshtastic applications must ensure they select a frequency band that aligns with their region [44]. Additionally, the device boasts an 850 mAh battery and a compact 1,54-inch SPI E-Paper Display, enhancing its utility and user experience [44]. Shipping times are roughly one month [44].

The Nano G1 Explorer, shown in Figure 3.5, introduces a key enhancement with its new internal wideband LoRa antenna, covering 815 MHz to 940 MHz frequencies [45]. This advancement, part of a significant update by B&Q Consulting, enables compatibility

Figure 3.4 LILYGO TTGO T-Echo dimensions [44]

with global LoRa frequency bands without altering the antenna, ensuring robust RF performance across different regions [45]. Additionally, attention was given to mitigate performance issues potentially caused by the human body [45]. The device also integrates an optional internal Li-Polymer battery charger and a buck-boost converter to maintain stable voltage, crucial for optimal RF circuit performance [45]. However, it's notable that shipping is restricted to a select group of countries including the US, Australia, Canada, and only a few European and Asian countries [45].

Figure 3.5 B&Q Consulting Nano G1 Explorer [45]

Taking an average 2023 United States dollar to Euro conversion rate of 0,9241 [46], the T-Echo costs approximately 50,28 EUR (54,41 USD [44]). The Nano G1 Explorer costs around 63,76 EUR (69 USD [45]), though this price does not include an internal battery, which must be supplied separately. It's important to note that these converted prices do not account for shipping costs or import taxes for Europe, which would significantly increase the final cost to the European end customer.

Despite the interesting features of these competitors, both devices have been out of stock for months, with no indication of these issues being resolved. This scarcity of ready-made Meshtastic communicators in Europe serves as the primary motivation for analyzing the manufacturing process of such devices.

The closest non-commercial, handmade device in terms of specifications and design is the TerraNode, shown in Figure 3.6, which costs EUR 118,79 [47]. Based on the information provided on the TerraNode webpage [47], we observe that it has similar specifications to our design, including a display, aftermarket antenna, and a large battery. However, upon further analysis, we note that the TerraNode uses an older and less power-efficient processor compared to our version [47]. The TerraNode is reasonably priced and still in production, while many other small-run handmade devices have been discontinued at various price points [47]. For comparison purposes, we will consider the TerraNode's price as is, since shipping from the UK to the EU would incur additional import taxes and shipping costs [47]. Due to the handmade nature of these types of Meshtastic devices, prices can vary significantly. This price of the TerraNode will be taken as the post-tax price achievable from the sale for our Meshtastic communicator design.

Figure 3.6 TerraNode Meshtastic device [47]

4. DEVELOPING PRODUCTION SIMULATION

Before building the simulation and its model, we must decide on the most appropriate type of simulation for our needs. We can immediately rule out a system dynamics simulation because the process we are modeling is not continuous. Different assembly stages can occur in parallel, especially with 3D printing being a mostly autonomous process.

Agent-based modeling could work for our needs, as we could have workstations, workers, and parts all be agents with their own desires and predetermined interactions. While this approach may give the most accurate and realistic simulation, it is also more computationally heavy. Emergent behavior is more of an issue, as it is very difficult to describe exact relationships between the different agents because all the agents have the same priority. In agent-based simulation, it's simple to create multiple instances of the same agent type, such as 100 printers and 100 workers, and have them interact with each other. However, it becomes challenging when you need to create agents that are similar but have slightly different behaviors or priorities.

For example, if you want to create 100 printers that are all slightly different or 100 workers who have different priorities, you would need to create a separate agent type for each variation. This is because it's difficult to modify similar agents after they've been created, and it's often better to make a new agent for each time they act differently. This can be a programming headache and may lead to a more complex and less manageable model.

The main issue for our situation with agent-based simulation is that the explored, affordable agent-based simulation software did not have a suitable time-tracking system, which would later allow us to see how much time has passed, the utilizations of people and machinery, and the average time taken for processes. This is because affordable agent-based modeling systems typically use a tick or turn system instead of a clock system. Instead of specifying that the simulation takes 100 seconds, you describe that the simulation takes 100 turns, and a turn is an action unit, not a time unit.

Thus, Discrete Event Simulation seems to be the best method for this project, as these software packages have a central system clock that drives the simulation. It also allows for the description of interactions between the systems in the model while assigning delays, queues, and priority schemes to manage the logic of the system. We will be modeling a discrete event simulation, which is stochastic because it incorporates

randomness, dynamic because it represents the model over time, and discrete because it is divided into discrete events rather than looking at changes continuously.

Next, we will examine and decide upon a suitable discrete event simulation modeling software.

4.1 Discrete event simulation software

A paper comparing open-source discrete event simulation software to commercial versions found that JaamSim is a favorite for novice users and recommended Salabim due to it being Python-based for advanced users, citing that Python offers more control [48].

However, the paper was not an exhaustive overview of open-source software for this purpose, as it seemed to focus more on comparing feature sets and usability. Thus, we'll have to make our own decision on which software to move forward with, based on our own research. The following are summaries of either affordable, free, or open-source discrete event simulation software packages.

Salabim

Salabim is a Python package designed for discrete event simulation, featuring integrated 2D animation that aids in validation and demonstration [49]. It is compatible across various platforms including Windows, Linux, OSX, and iOS, notably providing unique support for iPad use [49]. Its application spans multiple fields such as transportation research, manufacturing, mining, and hospital logistics [49]. Unlike SimPy, Salabim offers additional functionalities like animation, queue management, state monitoring, data collection, and statistical analysis tools [49]. Salabim's popularity is evident from the 2227 downloads it received in the last month on the PyPI package manager [50].

JaamSim

JaamSim is a widely used, Java-based, free, open-source discrete-event simulation software available since 2002, with over 10,000 downloads annually [51], [52]. It boasts user-friendly features like a drag-and-drop interface, interactive 3D graphics, and superior input and output processing, which ease the development of simulation

models [51], [52]. A standout feature of JaamSim is its customization capability, allowing users to create and integrate new object palettes tailor-made for their specific needs, enhancing model development efficiency by focusing on object logic over programming [51], [52]. Unlike other software, JaamSim uses standard Java for coding, enabling straightforward event- or process-oriented model logic coding via simple classes and methods [51], [52]. Its current version includes a diverse range of palettes for various simulation components, enhancing its versatility for different simulation requirements [51], [52].

OpenSIMPLY

OpenSIMPLY is a powerful, free, open-source simulation software compatible with Windows and Linux that excels in discrete-event simulations [53]. It supports an unlimited number of simulation events and includes executable demos, like queuing systems and call center models [53]. This software is accessible for those familiar with Delphi and Lazarus/Free Pascal, designed to simplify simulation processes requiring minimal coding, and comes with extensive educational resources [53]. OpenSIMPLY allows for both high-level and low-level simulation modeling, facilitating the creation of models with ease or the execution of more complex simulations efficiently [53]. Its notable abilities include flexible statistics collection, dynamic model adjustments during runtime, and suitability for a wide range of applications from scientific research to education, making it a versatile tool for traffic, network simulations, and more [53].

SimPy

SimPy is a simulation framework in Python, designed for event-driven process modeling [54]. Its first version was published in 2002 [55]. SimPy was originally based on ideas from Simula and Simscript, but uses standard Python [55]. It's useful for simulating the operations of systems, like how customers interact within a store, by using Python functions [54]. It contains tools for creating scenarios where resources are limited, such as checkout lines in a store [54]. You can run these simulations quickly, in real-time, or step by step [54]. However, SimPy is not ideal for continuous, non-interacting, or fixed-step simulations [54]. It comes bundled with learning resources like tutorials and is open-source under the MIT License [54]. Users are encouraged to share their experiences and techniques with the community through the SimPy mailing list [54].

SimPy's popularity is evident from the 6130 results returned on Google Scholar when searching for the keyword SimPy [56]. In the last month, it received 161078 downloads

on the PyPI package manager [57] and has a total of 55034 downloads on the Anaconda package manager [58]. Due to SimPy's popularity, there are many re-implementations of SimPy and libraries similar to SimPy such as SimSharp (C#), ConcurrentSim (Julia), and Simmer (R) [54].

Simmer

The simmer package introduces a high-level process-oriented modeling framework to the R language, incorporating the innovative concept of "trajectory" for defining standardized actions across similar process types [59]. This mechanism provides a streamlined and flexible modeling approach, enhanced by a robust C++ simulation core and an extensive R API [59]. Designed to simplify and optimize discrete event simulation within R, simmer allows for straightforward replication, parallelization, and data analysis through its automatic monitoring and flexible extension capabilities [59]. Despite potential challenges integrating R and C++, its performance competes well with equivalents like SimPy for Python and SimJulia for Julia [59]. Simmer's popularity is evident from the 2137 downloads it receives per month [60].

ConcurrentSim

ConcurrentSim, previously known as SimJulia, is a mature Julia package aimed at discrete event process-oriented simulation, drawing inspiration from SimPy, a similar Python library [61]. This toolkit, one of Julia's most established, is particularly beneficial for those familiar with Python's SimPy, though newcomers are encouraged to compare it through trial projects [61]. It leverages coroutines, facilitated by ResumableFunctions.jl, to navigate through time in simulations efficiently [61]. The development and maintenance of ConcurrentSim involve contributions from Ben Lauwens of the Royal Military Academy in Brussels, along with volunteers from JuliaDynamics and QuantumSavory [61].

4.1.1 Simulation software selection decision matrix

When choosing the best open-source discrete event simulation software, it's essential to systematically evaluate various options based on a set of crucial criteria. As detailed in the decision matrix in Table 4.1, these criteria include Versatility, Popularity, Ease of Setup, Functionality, and Support, each weighted according to its significance in

impacting user experience and practical utility. Popularity is given the highest weight in the decision matrix. It reflects the community support and reliability, which are pivotal for the software's long-term viability.

Table 4.1 Discrete event simulation software decision matrix

Criteria	Weight	Salabim	JaamSim	OpenSIMPLY	SimPy	Simmer	ConcurrentSim
Versatility	0,2	3,5	4	2,5	4,5	4	3
Popularity	0,3	3	4	1	5	4,5	3,5
Ease of Setup	0,2	3	3,5	4	4,5	3,5	2,5
Functionality	0,2	4	4,5	2	4	3,5	4
Support	0,1	3	5	1	4	4	2
Total Score		**3,35**	**4,3**	**2,15**	**4,5**	**3,85**	**3,1**

Versatility evaluates how well each software can integrate into different technological environments, a vital aspect for those looking to embed the software into varied infrastructures. Ease of Setup is particularly important, as a straightforward installation process is critical for user adoption and immediate productivity. Functionality is scored based on the range and depth of features available, determining the complexity and variety of simulations the software can handle. Support encompasses the availability and quality of guidance and community engagement, essential for effective problem-solving and learning.

From personal experiences: Salabim, while robust with features like 2D animation, presented some challenges during installation. JaamSim, ideal for intricate 3D modeling, provides excellent support, though the requirement of Java might be a hurdle for some. OpenSIMPLY, despite being user-friendly to set up, lags in community support and modernity. SimPy, on the other hand, offers an excellent balance across all dimensions. It especially excels in ease of use and widespread adoption within the Python community, making it a favorable choice for a variety of users. Simmer and ConcurrentSim offer specific benefits to users in statistical and Julia programming environments, respectively, but do not provide the same level of overall satisfaction as SimPy.

After careful consideration and combining both subjective experiences and the criteria in our decision matrix, SimPy emerges as the most advisable choice. Its strong performance across various essential aspects, particularly in community support and ease of integration into existing systems, affirm its suitability for engaging effectively with discrete event simulations.

4.2 Designing manufacturing layout and estimating manufacturing times

In this section, we will discuss the design of the manufacturing cell layout for the Meshtastic communicator and estimate the manufacturing times for each process. The layout design will be informed by lean manufacturing principles, such as 5S and cellular manufacturing, to optimize efficiency and productivity. We will also consider the application of continuous flow, one-piece flow, and the Kanban pull system to minimize waste and ensure a smooth production process.

4.2.1 Design of manufacturing cell

The design of the manufacturing cell for the Meshtastic communicator is informed by several lean tools, primarily 5S and cellular manufacturing. The application of 5S is evident in the separation of workstations into three distinct areas, ensuring that each workstation has all the necessary tools in place. This allows for efficient completion of tasks, as only the required items are taken, worked on, and then returned. The manufacturing cell follows a cellular layout, where resources and processes are grouped together to meet the specific processing needs of the Meshtastic communicator. This layout enables efficient flow and flexibility in adjusting the number of workers based on demand, starting with one worker and with the ability to scale up to a maximum of four.

The manufacturing process for the Meshtastic communicator can be classified as a batch process, as it involves producing multiple items simultaneously. However, the implementation of continuous flow and one-piece flow within the cell helps to minimize waste and improve agility, making the process more efficient and responsive to changes. One-piece flow is a crucial design decision for this small-scale manufacturing process, as designs can easily change, parts availabilities can vary, and the process needs to be agile.

Producing pieces to stock is avoided, as they may become obsolete with new product revisions, and it's a waste to produce to stock if it's not needed. This also highlights the importance of having the right number of printers to continuously feed the rest of the process.

The one-piece flow is closely related to the pull system, Kanban. A small amount of waste in the form of extra 3D printed parts is accepted to minimize the waste of operator waiting time. The pull system ensures that the operator only assembles products on demand, even if all the printers are ready with prepared parts. Key performance indicators, such as utilization, are monitored to ensure that workers are utilized to the best of their abilities and that there are no excess printers, which would also be a waste.

The manufacturing cell layout, as depicted in Figure 4.1, is designed as a U-shaped configuration with a floor plan measuring 4 meters wide by 3,6 meters long. The left side of the cell consists of three workstations: the computer station, the soldering station, and the assembly station. Each station measures 1 meter by 0,6 meters. The center of the U-shaped cell features a rack or shelves that can accommodate 3D printers on at least three levels, ensuring easy access for the operator to remove parts and perform maintenance. The right side of the cell is dedicated to storage, housing 3D printer filament, spare parts, and all the necessary components for assembling the communicator. This area also serves as a storage space for finished communicators, ready to be shipped out daily.

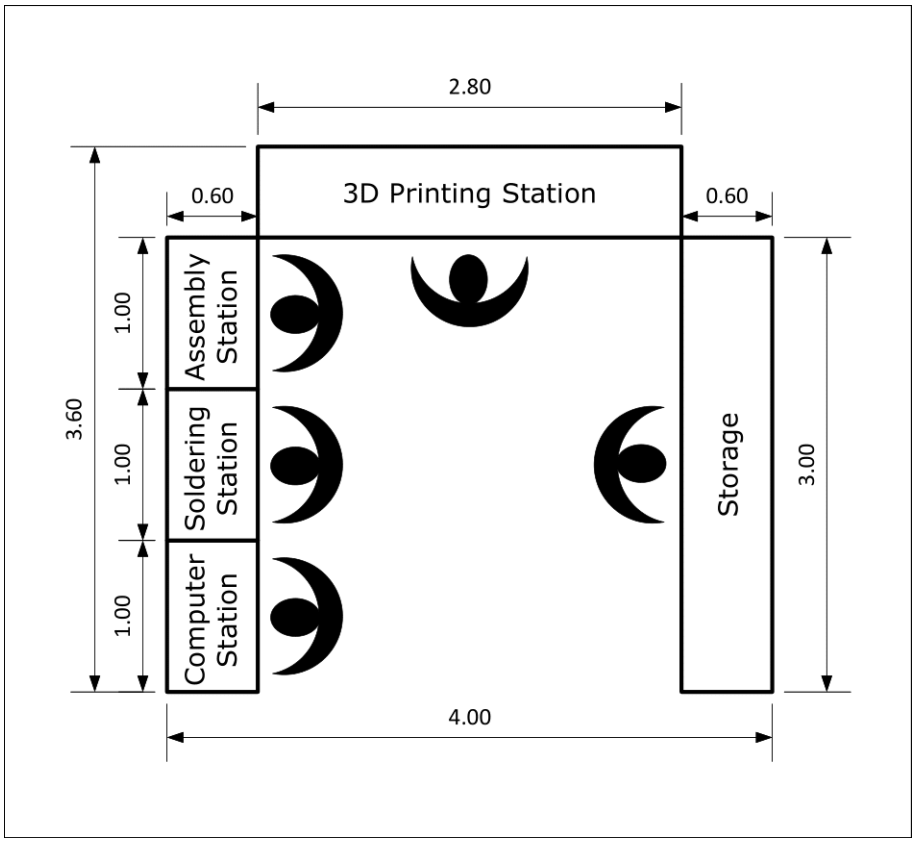

Figure 4.1 Layout of the Meshtastic communicator manufacturing cell (scale 1:50)

This layout design, informed by lean principles, provides a foundation for understanding the manufacturing process and the relationships between the various workstations. It also helps in determining the minimum cell size required for the manufacturing process, which can be useful when considering future expansions or modifications to the layout. The combination of the cellular layout, batch process, and lean principles creates an efficient and flexible manufacturing environment for the Meshtastic communicator, ensuring high productivity and responsiveness to changing requirements.

4.2.2 Estimation of manufacturing times

In this section, we will estimate the manufacturing times for each station involved in the production of the Meshtastic communicator. The time estimates provided in this section will serve as a foundation for simulating the manufacturing process and identifying potential areas for optimization. By understanding the time required for each task at the different stations, we can better plan the production workflow and allocate resources effectively.

Methodology for estimating manufacturing times

To estimate the manufacturing times for each workstation, we broke down the manufacturing process into manageable steps based on the bill of materials and the designed manufacturing cell layout. While actual time measurements were not available, we relied on the authors' and others' past experiences and best estimates to assign reasonable durations to each step. This approach allows for a more granular and adaptable estimation process, as individual step times can be updated as more data becomes available during the actual production phase. By providing detailed descriptions of the assembly processes and steps, we aim to make the time estimates transparent and open to scrutiny. It is important to note that these estimates are rough approximations and will likely differ from the actual values. To account for this uncertainty, variability will be introduced into the simulation model. The goal is to establish a solid foundation for estimating the production line's parameters, such as throughput and cost, prior to its actual implementation. The time estimates are based on the assumptions that the workers are familiar with the tasks and that all necessary tools and materials are readily available at the workstation, following the 5S methodology.

Computer station time estimation

The computer station is responsible for flashing the firmware onto the RAK4631 boards and preparing them for the subsequent assembly process. The input to this station is a set of RAK4631 and RAK19007 boards, along with the necessary antennas and label paper. The worker begins by snapping the two boards together and securing them with four M1,2 x 3 mm screws. Next, the Bluetooth antenna is permanently attached to the board, followed by a temporary LoRa antenna. It is crucial to attach the antennas before powering on the board, as failing to do so can result in damage to the board.

Once the antennas are in place, the worker connects the device to the computer via USB and ensures that the drive is mounted. The firmware, which has been previously downloaded to minimize the need for an internet connection during the flashing process, is then copied to the device. After the flashing is complete, the worker waits for the device to reboot, then removes the USB connection and the temporary LoRa antenna. Finally, the label for the device and the packaging is printed, preparing the device for the next stages of the assembly process.

Table 4.2 provides a detailed breakdown of the estimated time for each task in the computer station.

Table 4.2 Time estimates for computer station

Task	**Estimated time (seconds)**
Snapping the RAK4631 and RAK19007 boards together	10
Securing the boards with four M1,2 x 3 mm screws	30
Attaching Bluetooth antenna permanently	60
Attaching temporary LoRa antenna	60
Connecting USB	5
Ensuring drive is mounted	5
Copying UF2 file	30
Waiting for device to reboot after flashing	15
Removing USB	5
Removing temporary LoRa antenna	30
Printing label	60
Total estimated time per device	**310**

Soldering station time estimation

The soldering station is responsible for establishing the electrical connections required for the device to function properly. This process is divided into two main parts: soldering the power connections and soldering the display connections.

In the first part, the worker begins by turning on the soldering iron and preparing the power connection wires by stripping and cleaning them. Heat shrink tubing is then cut and slid onto the wires before soldering the power connections. After soldering, the heat shrink tubing is shrunk onto the connections to provide insulation and protection. It is important to note that the power connector is not snapped into the board at this point, and the power switch must be in the off position to prevent the device from being powered on before the antennas are attached.

The second part of the soldering process involves preparing the display connection wires and soldering them to the PCB. Heat shrink tubing is applied to the display cable for organization and protection.

Table 4.3 and Table 4.4 provide a detailed breakdown of the estimated time for each task in the soldering station.

Table 4.3 Time estimates for soldering power connections

Task	**Estimated time (seconds)**
Turn on soldering iron	5
Strip and clean power connection wires	60
Cut and slide heat shrink tubing for power connections	30
Solder power connections (3 × 45 seconds)	135
Shrink heat shrink tubing (3 × 12 seconds)	36
Total estimated time per device	**266 seconds**

Table 4.4 Time estimates for soldering display connections

Task	**Estimated time (seconds)**
Strip and clean display connection wires	120
Cut and slide heat shrink tubing for display cable	15
Solder display connections (8 × 30 seconds)	240
Shrink heat shrink tubing on display cable	12
Total estimated time per device	**387 seconds**

Assembly station times estimation

The assembly station is responsible for putting together all the components of the device and packaging it for distribution. The input to this station are the 3D printed case, PCB with display, battery assembly, and other necessary components.

The worker begins by mounting the radio PCB on the central 3D printed bracket using four M2,5 screws, followed by attaching the SMA antenna adapter to the radio board. The reset button cover is installed into the midframe, and the SMA antenna adapter is screwed into the frame. The SMA antenna is then attached, and the Bluetooth antenna, which is already connected to the board, is taped onto the frame.

Next, the display is mounted to the top plate using four M2 x 4 mm screws and washers, and the power switch is mounted to the mid-frame using two M2 screws. The battery holder is attached to the backplate using double-sided tape, and the JST-PH-2 power connector is connected to the mainboard, ensuring that the antenna is attached and the power switch is off.

Finally, the device is assembled using four M3 x 20 mm nuts and bolts, and a printed label is attached to the back cover. The device is then packaged in a cardboard box with eco-friendly filler to prevent movement during shipping, and labels are installed on the outer packaging to indicate the contents.

Table 4.5 presents the estimated average assembly times for each step of the process.

The time estimates provided in this section will serve as a foundation for simulating the manufacturing process and identifying potential areas for optimization. By understanding the time required for each task at the different stations, we can better plan the production workflow and allocate resources effectively.

Table 4.5 Estimated average assembly times for each step

Task	**Estimated time (seconds)**
Mount radio PCB using four M2,5 screws	45
Mount SMA antenna adapter to radio board	25
Install reset button cover into midframe	15
Screw SMA antenna adapter into frame	20
Attach SMA antenna	10
Tape Bluetooth antenna onto frame	30
Mount display using four M2 x 4 mm screws	60
Mount power switch to mid-frame using two M2 screws	30
Mount battery holder using double-sided tape	40
Connect JST-PH-2 power connector to mainboard	20
Assemble device using four M3 x 20 mm nuts and bolts	90
Attach printed label to back cover	25
Package device in cardboard box with eco-friendly filler	60
Install labels on outer packaging	20
Total estimated time per device	**490**

3D printing duration estimation

The selected 3D printer for this project is the Creality K1 Speedy 3D Printer, which utilizes Fused Deposition Modeling (FDM) technology (Figure 4.2) [62]. This printer offers a build volume of 220 mm x 220 mm x 250 mm, suitable for various printing needs, and boasts a printing speed of up to 600 mm/s [62]. The manufacturer provided specifications indicate an printing accuracy of 100 mm ± 0,1 mm and layer heights ranging from 0,1 mm to 0,35 mm [62]. The K1 Speedy's moderate size (355 mm x 355 mm x 480 mm) and weight (12,5 kg) make it a feasible option for professional settings, and it consumes 350W of power [62]. The printer comes with advanced features such as an auto-leveling system, a self-check mechanism, a ceramic heater capable of handling temperatures up to 300°C, and dual-fan technology to reduce model warping and stringing [62]. It also features a flexible, heat-resistant plate for easy model removal and is delivered fully assembled, ensuring user-friendliness. The printer is priced at 459 EUR [62].

Figure 4.2 Creality K1 Speedy 3D printer [62]

Creality Print, an FDM slicing software, is used to manage the 3D printing tasks efficiently [63]. This software allows users to operate their printers remotely from their computers, adjust various printing parameters, and interact directly with the Creality Cloud for model editing and G-code import/export [63]. It also offers connectivity options via USB or Wi-Fi and enables direct printing from the Creality Cloud APP, enhancing accessibility and user convenience.[63]

Figure 4.3 shows a G-code preview from the slicing software, displaying a detailed view of the 3D model segmented into multiple parts for printing. The image provides information on the total printing time (approximately 2 hours and 28 minutes), material weight (55,0821 grams), and material length (18,47 meters). The color-coded key correlates specific colors with the types of structures and their respective printing times and percentage contributions to the overall print. We estimate 60 seconds for removing the printed part, performing a quick visual inspection on the part and resetting the printer.

PET-G was chosen as the 3D printing filament for its ease of printing, minimal shrinkage and warping, durability, and the fact that it does not emit harmful fumes or require ventilation while printing [64]. Considering the cost of one kilogram of PETG filament (23,99 euros) [65] and the part weight (55,0821 grams), the material cost for the 3D printed part is approximately 1,33 euros.

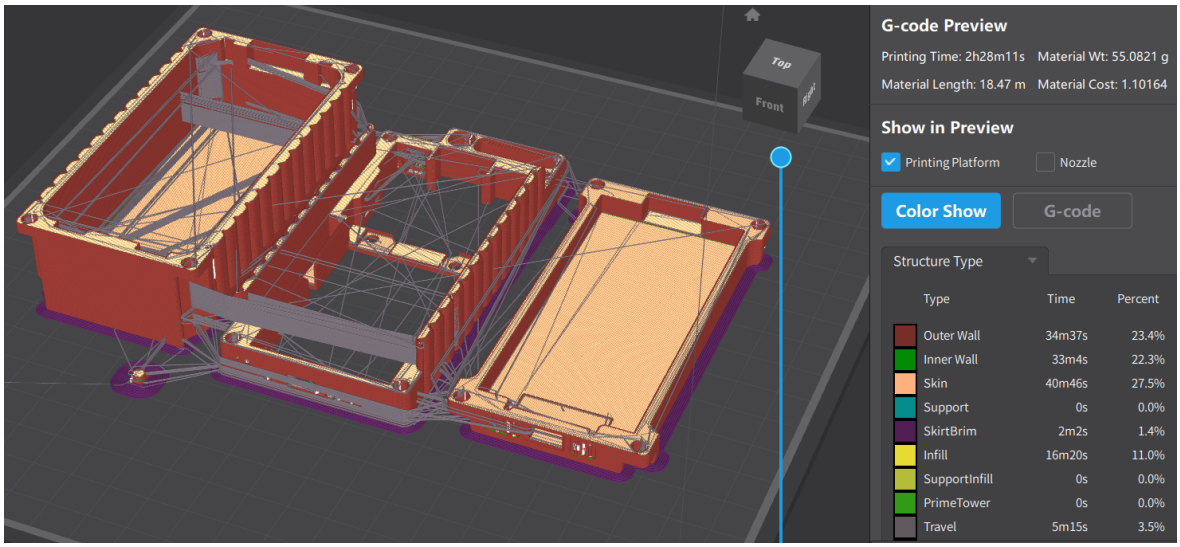

Figure 4.3 Creality Print Slicer program with estimated print time.

4.3 Designing and developing the simulation

In a previous chapter, we discussed the general elements of discrete event simulation models (see Table 2.3). These elements include create nodes, terminate nodes, activity nodes, entities, resources, paths, batch/unbatch nodes, and information links. Each of these elements plays a crucial role in representing the real-world system being simulated.

Now, let's examine how these elements are represented in the provided discrete event simulation code. Table 4.6 breaks down the specific elements present in this simulation model, describing their roles and interactions within the system.

As we can see from Table 4.6, the printers and the operator workflow process serve as the create nodes, generating parts and PCBs, respectively. The assembly process acts as the terminate node, where finished products leave the system. The activity nodes include the printers, flashing station, soldering station, and assembly station, each responsible for a specific task in the production process. The entities in this simulation are the parts, PCBs, and finished products, while the resources are the printers, various stations, and the worker. The paths describe the movement of parts and PCBs through the system, and the information links track work-in-progress and utilization metrics. By understanding how these elements are represented in the code, we can better comprehend the structure of the simulated system, and make informed decisions about potential improvements.

Table 4.6 Elements of discrete event simulation model for the code

Element	Description
Create node	The printers generate parts to enter the system. The operator workflow process generates PCBs for flashing.
Terminate node	The assembly process is the departure point for finished products leaving the system.
Activity nodes	Printers: Print parts. Flashing station: PCBs are flashed. Soldering station: PCBs are soldered. Assembly station: Parts and PCBs are assembled into products.
Entities	Printed parts, PCBs, and finished products.
Resources	Printers, flashing station, soldering station, assembly station, and the worker.
Paths	Parts move from printers to the assembly station. PCBs move from the flashing station to the soldering station and then to the assembly station.
Information links	Work-in-progress (WIP) counters for parts, flashed PCBs, and soldered PCBs. Utilization tracking variables for the worker, printers, flashing station, soldering station, and assembly station.

4.3.1 Determining simulation input parameters

Table 4.7 presents the input values for simulating a manufacturing process that involves 3D printing and manual assembly by a single operator. The process follows a one-piece flow manufacturing approach, with the operator moving lightweight parts (under 100 grams) between workstations arranged in a U-shape within a 4x4 meter floor space. The simulation aims to determine the optimal number of 3D printers required to keep the operator fully utilized. The times provided in the table represent the estimated values for the handling of 3D printed parts and processing times at each workstation. These estimated times will serve as the basis for the simulation, with variability to be added to better represent real-world conditions.

Table 4.7 Input values for the manufacturing process simulation

Parameter	**Estimated Time (seconds)**
Operator handling time for 3D printed parts	60
Workstation 1: Computer station	310
Workstation 2: Soldering station	653
Workstation 3: Assembly station	490
3D printer total printing time	8891

4.4 Description of simulation code and logic

Understanding the structure and logic of the simulation code is crucial for interpreting the results and making informed decisions about the manufacturing process. This section provides an overview of the Python code used to implement the discrete event simulation model discussed in this chapter. By examining the key components of the code and their interactions, readers can gain a deeper understanding of how the simulation represents the real-world system and how various elements work together to model the manufacturing process.

Appendix 1 presents the Python code for the discrete event simulation model discussed in this chapter. To better understand the code structure, let's take a closer look at its main components. Table 4.8 provides an overview of the simulation code, breaking it down into several key sections.

The code begins with an initialization section, where constants, variables, and the random number generator are defined. This section sets up the foundation for the simulation, including the number of printers, simulation time, and various task durations.

Next, the code defines task duration functions, which generate random durations for each task in the simulation, such as flashing, assembly, soldering, and printing. These functions use the constants defined in the initialization section and the random number generator to create realistic task durations.

Table 4.8 Simulation code overview

Section	Description
Initialization	Define constants, variables, and random number generator
Task duration functions	Define functions for generating randomized task durations
Printer process	Define the process for printers to produce parts
Operator workflow process	Define the process for the operator to perform tasks
Simulation setup	Create the simulation environment and processes
Simulation execution	Run the simulation until the specified time
Results and metrics	Print the simulation results and utilization metrics

The printer process and operator workflow process sections define how printers produce parts and how the operator performs tasks, respectively. The printer process describes how printers produce parts, while the operator workflow process outlines the sequence of tasks performed by the operator, including flashing PCBs, soldering PCBs, collecting parts from printers, and assembling products.

In the simulation setup section, the simulation environment is created, and the printer and operator workflow processes are initiated. The calculate start delay function is used to stagger the start times of the printers.

The simulation execution section runs the simulation until the specified time, using the env.run() function from the SimPy library.

Finally, the results and metrics section prints the simulation results, such as the total number of parts printed, PCBs flashed, PCBs soldered, and products assembled. It also calculates and displays utilization metrics for the worker, printers, flashing station, soldering station, and assembly station.

By understanding the structure of the code and the purpose of each section, readers can better comprehend how the discrete event simulation model is implemented and how the various elements discussed in Table 4.6 interact within the simulation.

The manufacturing process in the simulation model is represented by two key flowcharts: the Printer Flow in Figure 4.4 and the Operator Flow in Figure 4.5. These flowcharts illustrate the logic and decision-making processes of the printers and the operator.

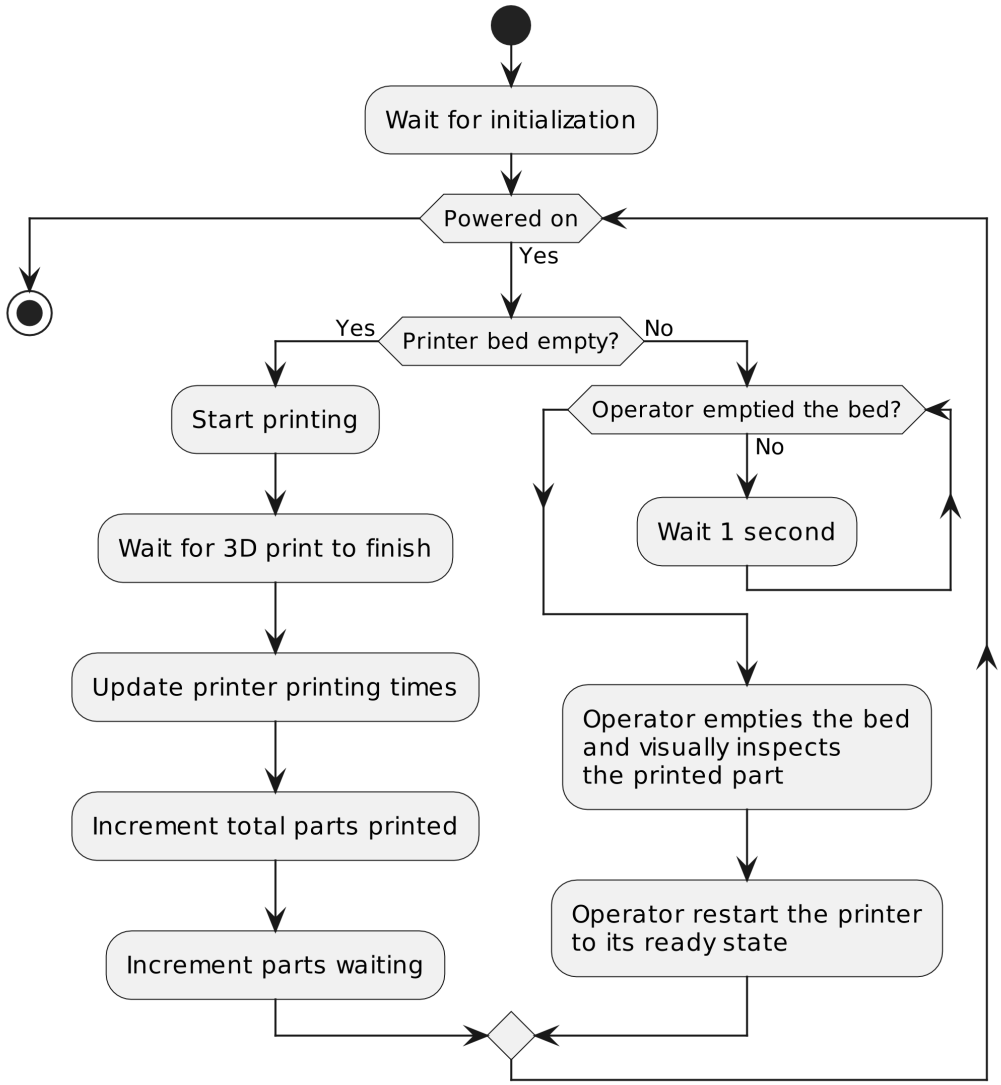

Figure 4.4 Single 3D printer operating logic flowchart

The Printer Flow (Figure 4.4) depicts the logic for each separate printer in the manufacturing cell. It is important to note that multiple printer logics can operate simultaneously and independently. Each printer follows a simple workflow: it checks if its print bed is empty, and if so, it starts printing a new part. The printer waits for the printing process to complete, updates its busy time, increments the total parts printed, and updates the print bed to full. If there is a part in it's print bed, then the printer remains idle and waits briefly before checking again.

The Operator Flow (Figure 4.5) represents the decision-making process of the operator in the manufacturing cell. The operator's primary goal is to assemble and package finished products, which requires both a soldered PCB and a printed part. The operator first checks if a soldered PCB is available. If one is available, the operator then checks if a printed part is also available. If both components are ready, the operator assembles and

packages the finished product, updates the relevant metrics, and increments the count of assembled products. If a soldered PCB is available but no printed part is ready, the operator waits for a printed part to become available. In the case where no soldered PCB is available, the operator moves on to the flashing and soldering process. The operator checks if a flashed PCB is available. If one is ready, the operator solders it, updates the soldering time and worker busy time, and increments the count of soldered PCBs. If no flashed PCB is available, the operator flashes a new PCB, updates the flashing time and worker busy time, and increments the count of flashed PCBs.

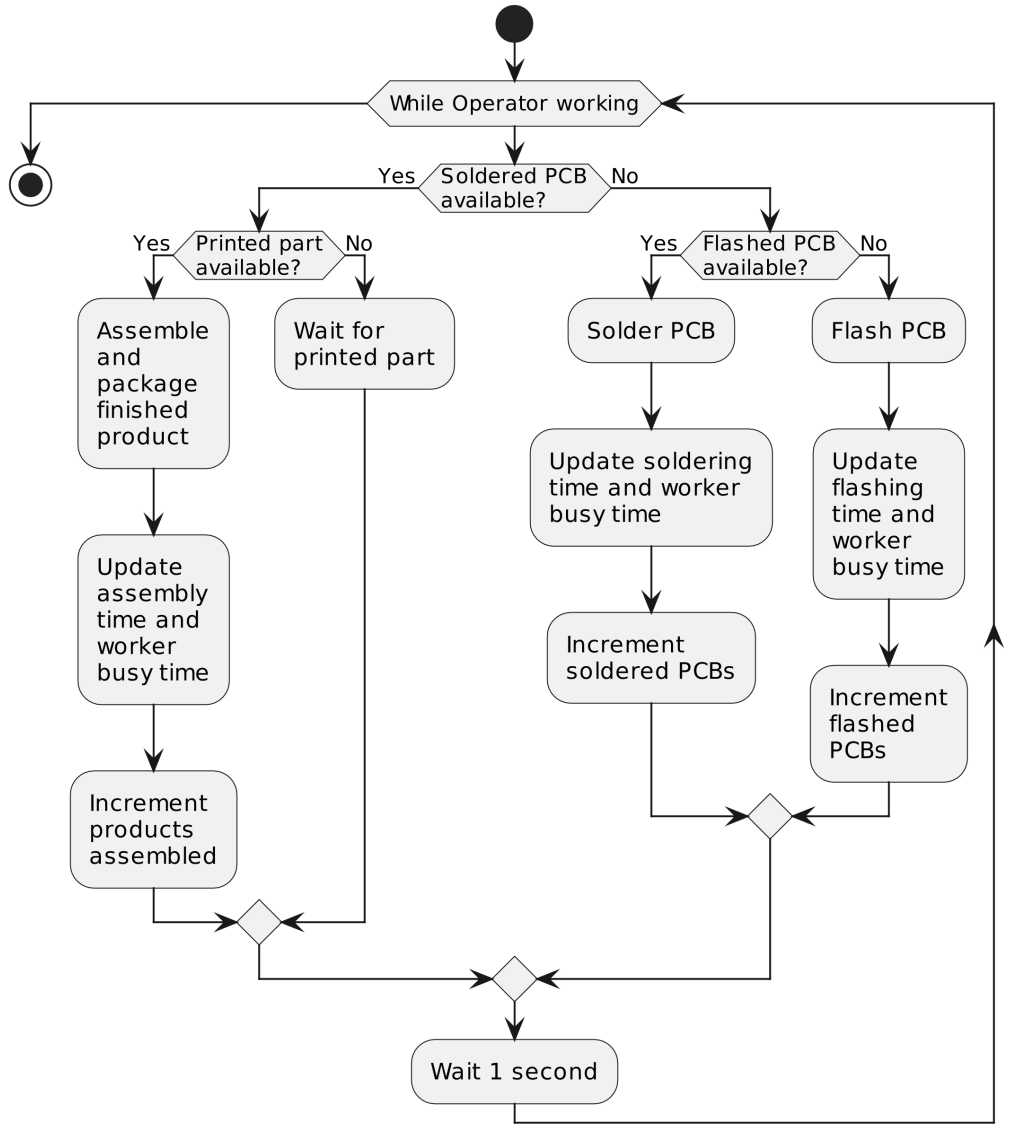

Figure 4.5 Operator workflow flowchart

After each action or decision, the operator waits briefly before checking the conditions again. This allows for the state of the system to change and prevents the operator from getting stuck in a loop. It is worth noting that while the current simulation model is

built for one operator in a single manufacturing cell, the logic can be easily extended to support multiple operators with minimal changes to the code. This scalability allows for the simulation of more complex manufacturing scenarios.

In the simulation of the production environment, the terminal output shown in Figure 4.6 quantifies the operational performance over 168 hours. The output displays key metrics such as the number of parts printed, PCBs flashed and soldered, and overall products assembled. These statistics are crucial for assessing the operational efficiency of the system. The terminal output also tracks the utilization percentages of different workstations and printers, providing insight into potential bottlenecks or underused resources. Monitoring these metrics helps in strategizing improvements, like adjusting the number of printers to align better with worker availability and demand, thus aiming to enhance worker utilization rates in future simulations.

```
Total Parts Printed: 198
Total PCBs Flashed: 199
Total PCBs Soldered: 199
Total Products Assembled: 198
Assembly Station Utilization: 16.10%
Flashing Station Utilization: 10.59%
Soldering Station Utilization: 21.41%
Worker Utilization: 50.08%
Printer 0 Utilization: 98.09%
Printer 1 Utilization: 98.27%
Printer 2 Utilization: 98.37%
Total Printers Utilization: 98.25%
Simulation took 5.29 seconds.
Average Assembly Time per Product: 3054.55 seconds
```

Figure 4.6 Terminal output of simulation

The debug output, as illustrated in Figure 4.7, offers a granular view of the discrete events within the simulation. Each entry logs specific events like the collection of parts from printers and the completion times of various production stages. This detailed report helps in verifying the sequential flow of operations and ensuring that there are no process overlaps that could lead to inefficiencies. Observing the debug output increases the time for simulation completion; however, the trade-off is invaluable for model validation. Understanding these timelines and dependencies is crucial for fine-tuning the system to reduce idle times and optimize the overall production flow. With analytics derived from the debug mode, potential improvements can be identified, contributing to the optimization of resource allocation and productivity.

```
Collected part from Printer 1 at time 600310.2434113041.
Printer 1 started at 600310.82
Product assembled at time 600801.5285831968. Total assembled: 197
PCB flashed at time 601111.7302662245. PCBs waiting for soldering: 1
PCB soldered at time 601769.7531261098. PCBs waiting for assembly: 1
Printer 2 finished at 603200.72. Parts waiting: 1
Collected part from Printer 2 at time 603260.7531261098.
Printer 2 started at 603261.72
Product assembled at time 603771.439948135. Total assembled: 198
```

Figure 4.7 Debug output of simulation

The dynamic simulations conducted vary based on whether event tracking (debug mode) is enabled. Without tracking, the system computes scenarios significantly faster—approximately 0,7 seconds. In contrast, with debug mode activated, the time increases to about 5,3 seconds due to the additional overhead of data logging. Despite this increased time, the depth of data collected when tracking is enabled proves essential for an accurate, thorough analysis and subsequent optimization of the system.

Development Environment

The simulation code was developed using Thonny, a Python IDE designed for learning and teaching programming, written by Aivar Annamaa at the University of Tartu [66]. Thonny's key advantages are its simplicity and ease of use, integrating necessary tools for programmers and reducing the learning curve for beginners [66]. Thonny is also free to use and open-source [66]. This makes it suitable for developing affordable simulations. Figure 4.8 shows a screenshot of the Thonny IDE with example code.

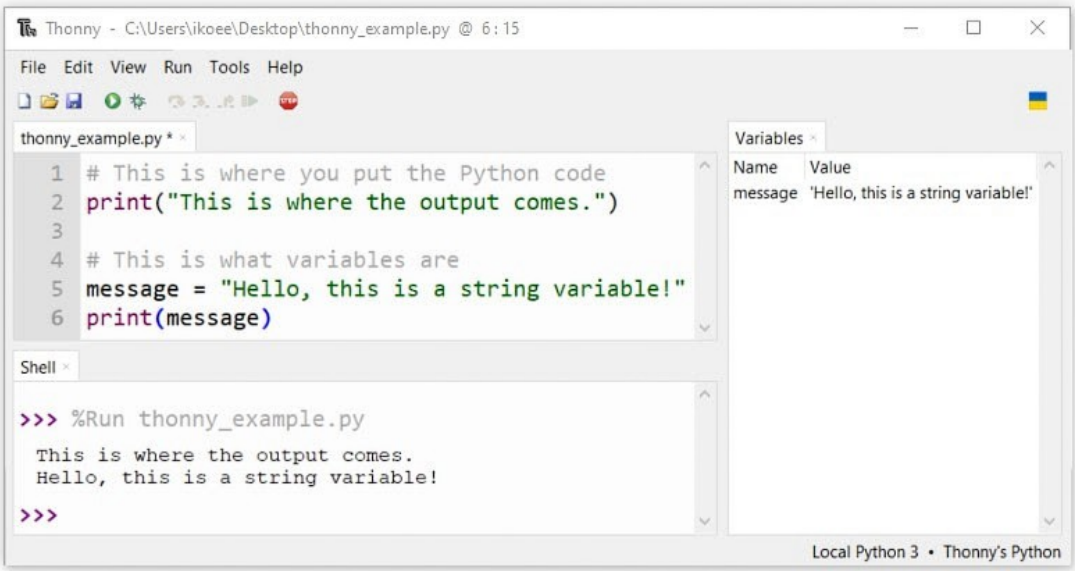

Figure 4.8 Thonny IDE with a short example script

4.4.1 Verification of Simulation Logic

Before introducing variability into the simulation, it is essential to verify the simulation logic to ensure that the model accurately represents the real-world system. This verification process involves testing the simulation with simplified parameters and analyzing the output to confirm that the logic is correct and consistent.

During the verification phase, the simulation was run using round whole numbers for process times and a limited number of printers, typically one or two. By varying these parameters and examining the output, it was possible to determine whether the simulation logic was functioning as intended.

The debug output played a crucial role in verifying the simulation logic. As shown in Figure 4.7, the debug output provided a detailed view of the discrete events within the simulation, including the collection of parts from printers and the completion times of various production stages. This granular information allowed for the verification of the sequential flow of operations and the identification of any logical inconsistencies, such as the operator performing multiple tasks simultaneously or printers restarting before being reset.

The terminal output, illustrated in Figure 4.6, offered a higher-level view of the simulation results, including key metrics such as the number of parts printed, PCBs flashed and soldered, and overall products assembled. By analyzing these metrics in conjunction with the debug output, it was possible to confirm that the simulation logic was accurate and that the model was behaving as expected.

Once confidence in the simulation logic was established through the verification process, more realistic variables and variability were introduced into the simulation. This gradual approach to building complexity ensured that the final simulation model was built on a solid foundation of verified logic, increasing the reliability and accuracy of the results.

4.4.2 Adding variability to the simulation

In order to make the simulation more realistic and representative of real-world scenarios, it is important to introduce variability into the input parameters. This section will discuss the use of three probability distributions - exponential, triangular, and uniform - to add variability to the simulation.

Exponential distribution

The truncated exponential distribution, available in the SciPy library, is a probability distribution that follows the exponential distribution but is limited to a specific range [67]. The probability density function of the truncated exponential distribution is defined by Equation 4.1, where (b) represents the shape parameter that determines the truncation point [67].

$$f(x, b) = \frac{exp(-x)}{1 - exp(-b)}, \quad \text{for } 0 \leq x \leq b, \tag{4.1}$$

In the simulation, we utilize the truncexpon function from the SciPy library to generate print times that follow the truncated exponential distribution [67]. We define a base print time and a maximum print time of double the base print time to set the range of the distribution, and a scale parameter to control the steepness of the drop-off [67].

To use the truncexpon function effectively, we need to calculate the shape parameter (b) [67]. This parameter determines the truncation point of the distribution [67]. We will calculate it using the Equation 4.2. The idea behind equation is that the max print time subtracted by the base print time represents the range of the distribution, i.e., the difference between the maximum and minimum possible print times [67]. Dividing this range by the scale parameter normalizes the range to a standardized scale [67]. The scale parameter controls the spread or dispersion of the distribution [67]. The resulting value of (b) represents the truncation point in terms of the standardized scale [67]. By calculating (b) in this way, we ensure that the generated print times are properly scaled and truncated according to the desired range and steepness.

$$b = \frac{\text{max print time} - \text{base print time}}{\text{scale}} \tag{4.2}$$

Inside the function that generates print times, we use truncexpon.rvs and Equation 4.2 to generate a random value from the truncated exponential distribution [67]. Finally, we add the generated value to the base print time to obtain the final print time that follows the truncated exponential distribution within the desired range. The function's work can be verified in Figure 4.9, which shows a plot of 10 thousand random samples.

By utilizing the truncated exponential distribution, we can generate print times that realistically simulate the behavior of a printing process, where the majority of print times are close to the base print time, but there is a decreasing probability of longer print times up to a maximum threshold. The decision to have the maximum print time as two times

the base print time was informed by the fact that, although unlikely, the print can fail near the end of the printing time.

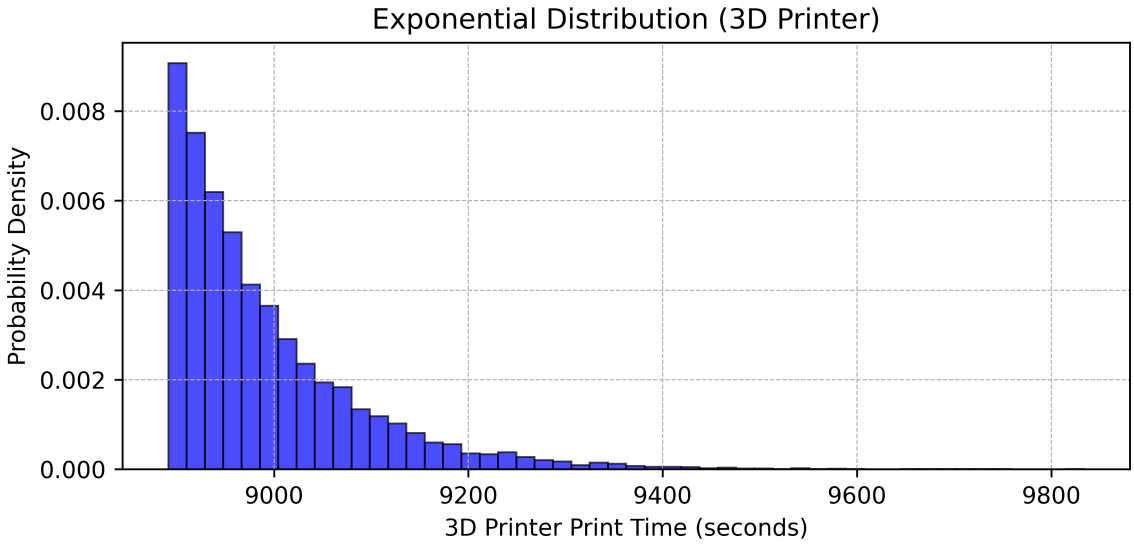

Figure 4.9 Exponential distribution plot

Triangular distribution

The triangular distribution is a continuous probability distribution characterized by three parameters: the minimum value (l), the maximum value (r), and the mode (m) [68]. The probability density function of the triangular distribution is defined by Equation 4.3, which consists of two linear segments joined at the mode [68].

In our code, we utilize the triangular distribution to generate random values within a specified range. The triangular distribution is particularly useful when we have limited information about the distribution of values, but we know the minimum, maximum, and most likely (mode) values [68].

To generate values from the triangular distribution, we use the numpy.random.triangular function [68]. This function takes three parameters: left, mode, and right, which correspond to the minimum value (l), the mode (m), and the maximum value (r) of the distribution, respectively [68].

By specifying these parameters, we define the range and shape of the triangular distribution. The generated values will be concentrated around the mode, with linear decreases in probability towards the minimum and maximum values [68].

Figure 4.10 illustrates the probability density function of the triangular distribution for 10 thousand random samples based on the specified parameters. The graph shows the

concentration of values around the mode and the linear slopes towards the minimum and maximum values.

$$P(x; l, m, r) = \begin{cases} \frac{2(x-l)}{(r-l)(m-l)} & \text{for } l \leq x \leq m, \\ \frac{2(r-x)}{(r-l)(r-m)} & \text{for } m < x \leq r, \\ 0 & \text{otherwise.} \end{cases} \tag{4.3}$$

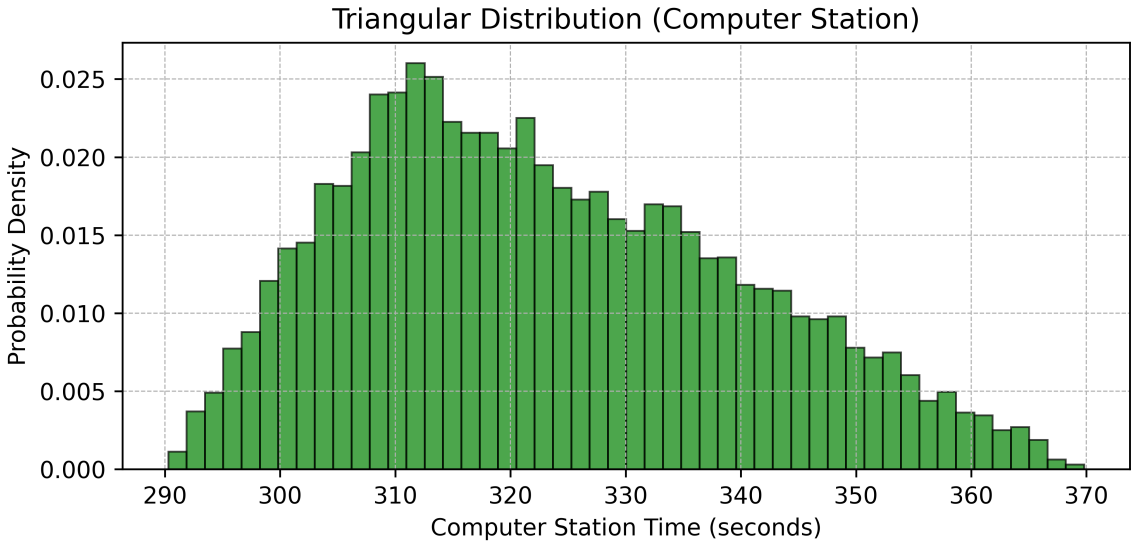

Figure 4.10 Triangular distribution plot

Uniform distribution

The uniform distribution is a probability distribution where all values within a given range have an equal probability of occurrence [69]. The probability density function of the uniform distribution is defined by Equation 4.4, where (a) and (b) represent the minimum and maximum values of the range, respectively [69].

In our code, we utilize the uniform distribution to generate random values uniformly distributed within a specified range. The uniform distribution is useful when we want to assign equal probabilities to all values within a given interval [69].

To generate values from the uniform distribution, we use the numpy.random.uniform() function [69]. This function takes two parameters: low and high, which correspond to the minimum value (a) and the maximum value (b) of the range, respectively [69].

By specifying the low and high parameters, we define the range within which the generated values will be uniformly distributed [69]. Each value within this range has

an equal probability of being selected [69].

Figure 4.11 illustrates the probability density function of the uniform distribution based on the specified range for 10 thousand random samples. The graph shows a constant probability density between the minimum and maximum values, indicating that all values within this range have an equal likelihood of occurrence.

$$p(x) = \frac{1}{b-a} \tag{4.4}$$

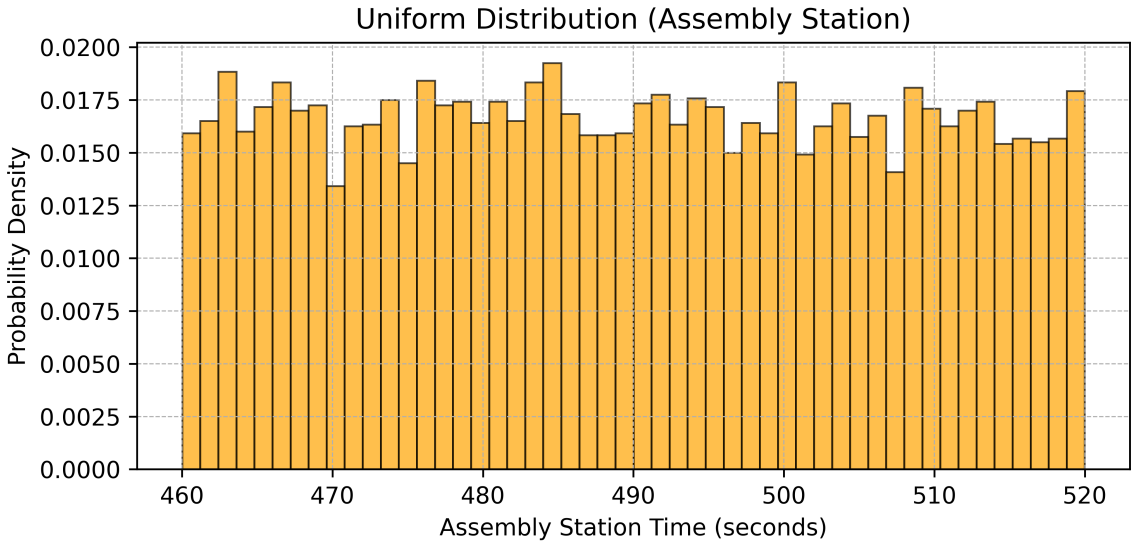

Figure 4.11 Uniform distribution plot

4.5 Analysis of simulation output and optimization of manufacturing setup

The discrete event simulation model developed in the previous sections allows us to analyze the performance of the manufacturing setup under various scenarios. By running simulations with different numbers of 3D printers, we can determine the optimal configuration that maximizes productivity while considering factors such as worker utilization and the need for printer maintenance. In this section, we will present the results of these simulations and discuss the implications for optimizing the manufacturing setup.

To determine the optimal number of 3D printers for our manufacturing unit, we conducted a series of simulations varying the number of printers from 1 to 10. The simulations assessed the total number of products assembled, the utilization of the assembly, flashing, and soldering stations, as well as the worker and total printer utilization. The results of these simulations are presented in Table 4.9 and Figures 4.12 and 4.13.

Table 4.9 presents the key metrics for each simulation run, including the number of products assembled, station utilizations, worker utilization, and total printer utilization. This table allows for a direct comparison of the performance of the assembly unit with different numbers of 3D printers.

Table 4.9 Simulation results for varying number of 3D printers

Number of Printers Used	Number of Products Produced	Assembly Workstation Utilization (%)	Computer Workstation Utilization (%)	Soldering Workstation Utilization (%)	Worker Utilization (%)	Average Printer Utilization (%)
1	66	5,34	3,56	7,15	16,70	97,97
2	132	10,70	7,04	14,28	33,34	98,20
3	198	16,10	10,59	21,41	50,08	97,98
4	264	21,48	14,12	28,40	66,63	98,11
5	329	26,77	17,57	35,44	83,04	97,83
6	387	31,51	20,72	41,83	97,91	96,13
7	390	31,75	20,87	42,15	98,64	83,23
8	390	31,75	20,87	42,15	98,64	73,08
9	390	31,75	20,87	42,15	98,64	65,13
10	390	31,75	20,87	42,15	98,64	58,76

Figure 4.12 shows the relationship between the number of printers and the total products assembled. This graph helps visualize the point at which adding more printers does not significantly increase output.

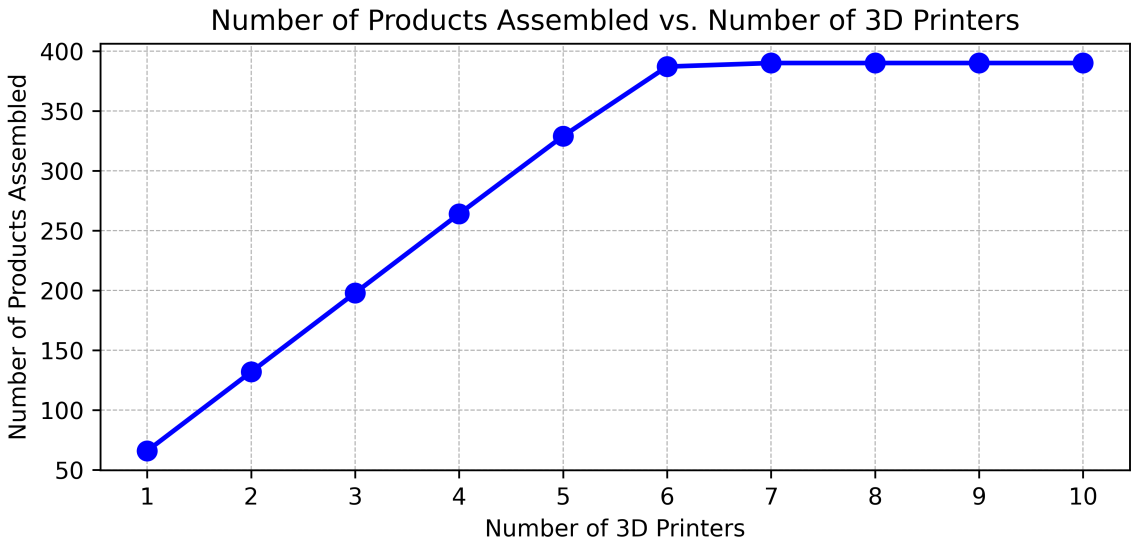

Figure 4.12 Relation between number of products assembled and number of printers in system

Figure 4.13 displays the utilization percentages of the assembly, flashing, and soldering stations, as well as the worker utilization and total printer utilization for each number of printers. This graph helps identify bottlenecks and the point at which the worker and printers reach their maximum utilization.

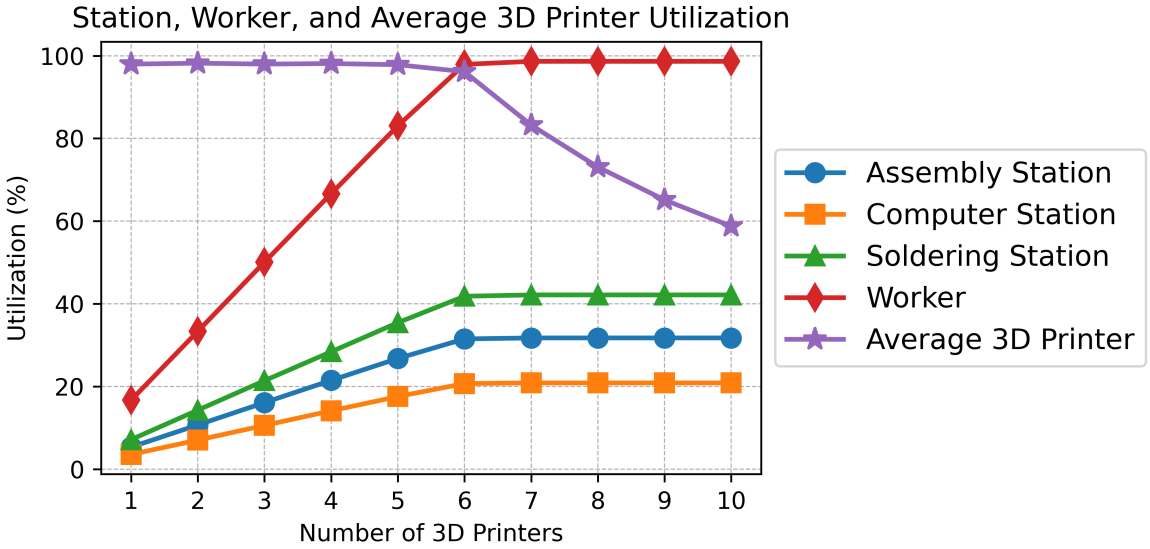

Figure 4.13 Relation between number of printers in system and average printer utilization, average worker utilization, and average workstation utilization

Based on the simulation results, it appears that the performance of the assembly unit converges around six printers. At this point, the worker utilization reaches 97,91 %, and the total printer utilization is at 96,13 %. Adding more printers beyond this point does not significantly increase the number of products assembled and leads to lower

printer utilization. However, it is important to consider the need for maintenance and service of the printers. Running the printers at very high utilization constantly may lead to increased wear and tear, resulting in more frequent breakdowns and maintenance requirements. Therefore, it may be beneficial to have an additional printer to allow for servicing without significantly impacting production.

Considering the simulation results and the need for printer maintenance, we recommend having seven 3D printers in the assembly unit. This configuration allows for near-maximum utilization of the worker and printers while providing a buffer for printer servicing. With seven printers, the assembly unit can produce 390 products per month with a worker utilization of 98,64 % and an average printer utilization of 83,23 %. The additional printer ensures that production can be maintained even when one printer is taken out for service, thus minimizing the impact on overall output.

4.6 Analysis of simulation development

Developing the simulation using Python and SimPy presented a unique experience compared to using fully-fledged solutions like Siemens Tecnomatix or Arena. While the latter offers comprehensive features for development, analysis, and setup, Python and SimPy provide a more streamlined and focused approach.

SimPy, which can be viewed as a a scheduling tool for parallel processes, enables faster development by simplifying the management of timings and dependencies. Python's simplicity is advantageous for creating simpler simulations, as it allows developers to focus on the essential aspects without being overwhelmed by excessive statistics and features. This intentional approach encourages developers to clearly define the simulation's objectives and desired outcomes.

Python's open nature and compatibility with version control tools facilitate collaboration and adaptability, allowing multiple people to work simultaneously on the same code and simulation. This eliminates the need for multiple licenses and reduces documentation time, as the working code itself serves as the primary source of documentation. In contrast, graphical-oriented simulation programs like Tecnomatix and Arena often hide variables in menus and submenus, making it challenging to transfer skills between platforms and requiring extensive documentation for others to run the simulation.

Developing everything in Python necessitates working in smaller operations, conducting frequent testing, and performing sanity checks. This approach promotes good coding practices and ensures that the simulation accurately represents the real-world system. By writing the code from scratch, developers gain a deeper understanding of the simulation's logic and can identify potential issues early in the development process.

The choice to use Python and SimPy aligns with the principles of Agile software development, which emphasizes flexibility, collaboration, and iterative development. The lightweight nature of Python simulations allows for quick iterations and re-running when needed, facilitating the Agile approach. It also aligns well with the eight lean wastes in the digital age, particularly in terms of precise data gathering. By collecting only the necessary data, simulations can run faster, consume fewer resources, and generate less data to store. This approach reduces waste and promotes efficiency, which is a core principle of Lean software development.

While Arena or Tecnomatix may be more accessible for those without a programming background, individuals with programming experience can benefit from the granular control and iterative development process offered by Python and SimPy. The choice to use open-source alternatives like SimPy and Python aligns with the agile and digital waste paradigm, promoting transparency, efficiency, and rapid iteration.

In conclusion, my personal experience with using Python and SimPy for simulation development has been positive, offering a more focused and efficient approach compared to fully-featured commercial solutions. The alignment with Agile principles and the eight lean wastes in the digital age, as discussed in the literature review, further reinforces the benefits of using these open-source tools for simulation development.

4.7 Affordable simulation guide for micro enterprises

This section presents a condensed guide for micro-enterprises and startups to evaluate their current or potential manufacturing processes and associated costs using affordable simulation techniques. It is important to note that much of the simulation work involves preparing for the simulation, not just the simulation itself. The goal is to condense the knowledge gained from this work into a practical, step-by-step approach that others can follow to benefit from simulation in their manufacturing processes.

Based on the experiences and insights gained from developing the Meshtastic communicator production simulation, this guide is designed to help micro-enterprises overcome the barriers to adopting simulation, such as limited resources, lack of expertise, and perceived complexity. By providing a structured, easy-to-follow approach, the guide aims to empower micro-enterprises to leverage simulation for improving their manufacturing processes, optimizing resource allocation, and making informed decisions about production planning and capacity expansion.

The guide emphasizes the use of open-source software, such as SimPy, to minimize costs and ensure accessibility. This focus on open-source tools aligns with the guide's pragmatic approach to simulation, encouraging users to start with a minimal viable model and gradually add complexity as needed, rather than striving for perfect realism from the outset.

Appendix 2 provides a detailed checklist that guides users through the process of implementing affordable simulation, which is summarized below:

1. Determine if simulation is necessary.
2. Identify the appropriate type of simulation.
3. Select a suitable open-source discrete event simulation software.
4. Design your manufacturing layout and estimate process times.
5. Develop the simulation model.
6. Analyze simulation output and optimize manufacturing setup.

This guide serves as a valuable starting point for micro-enterprises and startups looking to leverage simulation for informed decision-making and continuous improvement of their manufacturing processes. By following this affordable simulation guide, micro-enterprises and startups can gain a better understanding of their manufacturing processes, identify areas for improvement, and make informed decisions about resource allocation, production planning, and capacity expansion. It is important to note that while this guide offers a foundation for implementing affordable simulation techniques, the actual impact and effectiveness may vary depending on the unique characteristics and challenges of each organization. Nonetheless, by starting with a minimal viable model and gradually incorporating new insights and data, micro-enterprises and startups can explore the potential benefits of simulation for their specific context.

5. MESHTASTIC COMMUNICATOR PRODUCTION FINANCIALS

The financial analysis of the Meshtastic Communicator manufacturing project is crucial for determining the viability and potential profitability of the venture. This chapter will discuss the estimated manufacturing setup costs, recurring costs, and the overall financial viability of the project.

5.1 Estimating manufacturing setup costs

The setup costs for the Meshtastic Communicator manufacturing include the necessary equipment, tools, and furniture required to establish a functional production environment. As shown in Table 5.1, the costs are divided into four main categories: soldering workstation, assembly workstation, computer workstation, and 3D printing.

The soldering and assembly workstations are equipped with essential tools for the assembly of electronic components, wiring, and final device assembly. The computer workstation includes a laptop for firmware flashing and a printer for documentation and labels. Furniture costs cover chairs, tables, and shelves to provide adequate workspaces and storage for components, tools, and finished products.

The 3D printing section constitutes a significant portion of the setup costs, with seven 3D printers required to produce the custom-designed case for the Meshtastic Communicator. The number of printers was determined through a detailed analysis in the previous chapter to ensure sufficient production capacity. The case design cost is also included in this section, estimated at 500 euros, as using the case design we used for manufacturing estimation would require obtaining a license for it.

The total estimated setup cost for the Meshtastic Communicator manufacturing is 5203 euros. This initial investment is crucial for establishing the necessary infrastructure to produce the device efficiently and effectively.

Table 5.1 Setup costs for Meshtastic communicator manufacturing

Description	Quantity	Cost (EUR)
Soldering Workstation		
Soldering Iron	1	100
Heat Gun	1	50
Soldering Jigs	1	30
Heat Shrink Holder	1	15
Assembly Workstation		
Screwdriver Set	1	30
Computer Workstation		
Laptop	1	500
Laser Printer (Black & White)	1	100
USB-C Cable	1	10
Rubber Pad	1	5
Furniture		
Chairs	3	150
Tables	3	300
Shelves	2	200
3D Printing		
3D Printers	7	3213
3D CAD Model Design	1	500
Total Setup Cost		**5203**

5.2 Estimating manufacturing recurring costs

To estimate the monthly recurring costs for the Meshtastic Communicator manufacturing, we consider the salary of one full-time employee and the cost of renting the required space, including utilities. The average salary of a solderer in Harjumaa, Estonia, in the fourth quarter of 2023 was used as a reference, as solderers have a higher base salary compared to other assembly workers, and the required skills align with the needs of this project [70]. In addition to the gross salary, the employer must pay a 33% social tax and a 0.8% unemployment insurance tax of gross salary [71].

The rent for the business property in Tallinn, Estonia, was estimated based on the average rent per square meter for business properties at the end of 2023, which was 10,5 euros [72]. The required space for the manufacturing setup is approximately 14,4 square meters. However, finding a suitable rental property for such a small area may prove challenging. Sub-renting from a larger space could be an option, but it may still be difficult to secure. Utilities are estimated at 50% of the rental cost, considering the high energy costs in Estonia and the additional energy consumption of the 3D printers, soldering iron, and other necessary equipment.

The monthly recurring costs are summarized in Table 5.2, which includes the employee salary, social tax, unemployment insurance tax, rent, and utilities. This sums up to be 2682,03 euros a month.

Table 5.2 Monthly recurring costs for Meshtastic communicator manufacturing

Description	**Cost (EUR)**
Employee salary (Gross)	1835,00
Social tax (33% of salary)	605,55
Unemployment insurance tax (0,8% salary)	14,68
Rent (14,4 m^2 at 10,5 EUR/m^2)	151,20
Utilities (50% of Rent)	75,60
Total Monthly Recurring Costs	**2682,03**

5.3 Estimating financial viability of manufacturing

The financial analysis of the Meshtastic communicator manufacturing project is crucial for determining the viability and potential profitability of the venture. To assess the project's financial aspects, we will consider the setup costs, recurring costs, and the estimated sale price of the device.

The payback period is a useful metric for evaluating the time required to recover the initial investment in a project [73]. It is calculated by dividing the cost of investment by the average annual cash flow, as shown in Equation 5.1 [73]. The break-even point, on the other hand, represents the point at which total revenue equals total costs, and it is calculated by dividing the fixed costs by the difference between the price per unit and the variable cost per unit, as shown in Equation 5.2 [74].

$$\text{Payback Period} = \frac{\text{Cost of Investment}}{\text{Average Annual Cash Flow}} \tag{5.1}$$

$$\text{Break-Even Point} = \frac{\text{Fixed Costs}}{\text{Price Per Unit} - \text{Variable Cost Per Unit}} \tag{5.2}$$

The setup costs, as shown in Table 5.1, amount to 5203 euros. This initial investment covers the necessary equipment, tools, furniture, and design costs required to establish the manufacturing infrastructure. The recurring costs, presented in Table 5.2, total 2682,03 euros per month. These costs include the employee salary, social tax, unemployment insurance tax, rent, and utilities. The Bill of Materials in Table 3.1 shows that the material cost for producing a single Meshtastic Communicator unit is 79,93 euros. The estimated sale price after taxes is 118,79 euros per unit, which is taken from an equivalent competitor after tax [47].

Using the base formulas for payback period (Equation 5.1) and break-even point (Equation 5.2), we can develop the three formulas specific to our financial analysis.

Equation 5.3 adapts the base payback period formula (Equation 5.1) to our specific case. We consider the setup costs as the cost of investment and the maximum potential monthly profit as the denominator, which is calculated by subtracting the material cost per unit and recurring cost per unit from the revenue per unit and multiplying the result by the maximum monthly production.

$$\begin{aligned} PP &= \frac{SC}{(RU - MCU - RCU) \times MMP} \\ \text{where: } PP &= \text{Payback Period} \\ SC &= \text{Setup Costs} \\ RU &= \text{Revenue per Unit} \\ MCU &= \text{Material Cost per Unit} \\ RCU &= \text{Recurring Cost per Unit} \\ MMP &= \text{Maximum Monthly Production} \end{aligned} \tag{5.3}$$

Equation 5.4 is a formula we developed to calculate the potential profit when operating at maximum capacity. It considers the revenue per unit, material cost per unit, recurring cost per unit, and the maximum monthly production.

$$
\begin{aligned}
PMC &= (RU - MCU - RCU) \times MMP \\
\text{where: } PMC &= \text{Profit at Max Capacity} \\
RU &= \text{Revenue per Unit} \\
MCU &= \text{Material Cost per Unit} \\
RCU &= \text{Recurring Cost per Unit} \\
MMP &= \text{Maximum Monthly Production}
\end{aligned}
\tag{5.4}
$$

Equation 5.5 is derived from the base break-even point formula (Equation 5.2). We consider the recurring costs as the fixed costs and the difference between the revenue per unit and material cost per unit as the contribution margin. This formula calculates the minimum number of units that need to be sold each month to cover the recurring costs.

$$
\text{Minimum Units} = \frac{\text{Recurring Costs}}{\text{Revenue per Unit} - \text{Material Cost per Unit}} \tag{5.5}
$$

The financial analysis, presented in Table 5.3, applies these formulas to assess the viability of the Meshtastic communicator manufacturing project. Since the data in the table is based on estimation and simulation, the output values discussed in this section will be rounded to four significant digits. The payback period (Equation 5.3) is approximately 12,51 days, assuming the maximum monthly production of 390 units. The potential profit at maximum capacity (Equation 5.4) is 12470 euros per month, indicating a significant return on investment. To cover the recurring costs, the project must sell a minimum of 70 (rounded up from 69,02 to hole numbers) units per month (Equation 5.5), which represents the breakeven point. The estimated setup costs are 5203 euros, and the recurring costs per month are 2682 euros.

Based on the financial analysis, the Meshtastic Communicator manufacturing project appears to be a promising venture with a relatively short payback period and a high potential for profitability. However, it is essential to consider other factors such as market demand, competition, and potential risks before making a final decision on whether to proceed with the project.

Table 5.3 Financial analysis of Meshtastic communicator manufacturing

Description	**Value**
Setup costs (EUR)	5203
Recurring costs per month (EUR)	2682,03
Material cost per unit (EUR)	79,93
Estimated sale price after taxes (EUR)	118,79
Maximum monthly production (Units)	390
Payback period (days)	12,51
Profit at maximum capacity (EUR/Month)	12473,37
Minimum units to cover recurring costs (Units/Month)	69,02

SUMMARY

This thesis evaluates the financial feasibility of manufacturing the Meshtastic communicator, an open-source, encrypted wireless mesh network device, within the context of a hypothetical one-employee micro-company in Estonia. The study focuses on estimating production costs, proposing a lean manufacturing layout, and using simulation to calculate manufacturing costs and profitability.

The research process involved exploring various open-source simulation solutions and selecting SimPy, a Python library, as the most suitable tool for developing a discrete event simulation model. The importance of simulation in this work cannot be overstated, as it enables the execution of complex cost calculations and provides crucial insights into the production capacity of a single worker.

The simulation model was designed to optimize the manufacturing process and estimate production quantities, considering factors such as the number of 3D printers and worker utilization. The results showed that six to seven 3D printers were optimal for maximizing productivity while allowing for smooth maintenance. With seven printers, the production unit can produce 390 products per month, with a worker utilization of 98,64 % and an average printer utilization of 83,23 %.

The financial analysis revealed promising results for the feasibility of producing the Meshtastic Communicator. The parts cost per unit is 79,93 euros, with manufacturing setup costs totaling 5203 euros and monthly recurring costs amounting to 2682 euros. With a sale price of 118,79 euros per unit and a maximum monthly production of 390 units, the payback period is 12,51 days, and the potential monthly profit at full capacity is 12470 euros. The project must sell at least 70 units monthly to cover recurring costs.

The author found the experience of developing with SimPy to be a positive one, as it offered basic functionality while allowing for more focused and efficient work. The use of an open-source and flexible toolkit also aligned with the principles of Agile development and avoiding the eight wastes of the digital age.

The results of this work demonstrate that using affordable, open-source simulation tools allows micro-enterprises to quickly assess the feasibility of production ideas. By using these tools, startups can iterate faster, avoid investing in unprofitable ventures, and focus on innovative, competitive products.

Furthermore, this thesis presents an affordable simulation guide for micro-enterprises and startups to evaluate their current or potential manufacturing processes and associated costs. The guide condenses the knowledge gained from the Meshtastic communicator production simulation into a practical, step-by-step approach that others can follow. It emphasizes the use of open-source software, such as SimPy, to minimize costs and ensure accessibility. The guide serves as a valuable starting point for micro-enterprises and startups looking to leverage simulation for informed decision-making and continuous improvement of their manufacturing processes.

In conclusion, this thesis offers a new strategic tool for small and micro-enterprises. The presented methodology can help micro-enterprises navigate the complexities of product development with greater ease and lower financial risk, potentially enhancing their competitiveness in the market. Future research could explore the application of agent-based simulation models, market demand analysis, product life cycle analysis, and project scalability.

KOKKUVÕTE

Selle magistritöö eesmärk on hinnata Meshtastic sidevahendi tootmise finantsilist teostatavust Eestis ühe töötajaga mikroettevõtte kontekstis. Meshtastic on avatud lähtekoodiga krüpteeritud traadita võrguseade. Uuringus keskendutakse tootmiskulude hindamisele, kuluefektiivse tootmisplaani väljatöötamisele ning simulatsiooni kasutamisele tootmiskulude ja kasumlikkuse arvutamisel.

Uurimisprotsessis uuriti erinevaid avatud lähtekoodiga simulatsioonilahendusi ning valiti SimPy, Pythoni teek, mis osutus kõige sobivamaks tööriistaks diskreetse sündmuse simulatsioonimudeli loomisel. Simulatsioon mängib selles töös väga olulist rolli, võimaldades teostada keerukaid kuluarvutusi ja andes väärtuslikku teavet ühe töötaja tootmisvõimekuse kohta.

Simulatsioonimudel loodi tootmisprotsessi optimeerimiseks ja tootmiskoguste hindamiseks, arvestades selliseid tegureid nagu 3D-printerite arv ja töötaja hõivatus. Tulemused näitasid, et optimaalne printerite arv tootlikkuse maksimeerimiseks ja sujuva hoolduse tagamiseks on kuus kuni seitse. Seitsme printeriga suudab tootmisüksus valmistada 390 toodet kuus, kusjuures töötaja hõivatus on 98,64 % ja printerite keskmine hõivatus 83,23 %.

Finantsanalüüs andis paljulubavaid tulemusi Meshtastic sidevahendi tootmise teostatavuse osas. Ühe toote osade maksumus on 79,93 eurot, tootmise alustamiskulud kokku 5203 eurot ja igakuised püsikulud 2682 eurot. Müües toodet hinnaga 118,79 eurot ja saavutades maksimaalse igakuise toodangu 390 ühikut, on tasuvusaeg 12,51 päeva ning võimalik igakuine kasum täisvõimsusel 12470 eurot. Püsikulude katmiseks peab ettevõte müüma vähemalt 70 ühikut kuus.

Autor leidis, et SimPy-ga arendamine oli meeldiv kogemus, kuna see pakkus põhilist funktsionaalsust, kuid võimaldas fokusseeritumat ja tõhusamat tööd võrreldes kommertslahendustega. Avatud lähtekoodiga ja paindliku tööriistakomplekti kasutamine oli kooskõlas ka Agiilse arenduse põhimõtetega ja aitas vältida kaheksat digitaalset raiskamist.

Selle töö tulemused näitavad, et kulusäästlike avatud lähtekoodiga simulatsioonitööriistade kasutamine võimaldab mikroettevõtetel kiiresti hinnata tootmisideede teostatavust. Neid tööriistu kasutades saavad iduettevõtted kiiremini itereerida, vältida investeerimist kahjumlikesse ettevõtmistesse ning keskenduda uuenduslikele ja konkurentsivõimelistele toodetele.

Lisaks esitab see magistritöö kulusäästliku simulatsioonijuhendi mikroettevõtetele ja iduettevõtetele nende praeguste või potentsiaalsete tootmisprotsesside ning seotud kulude hindamiseks. Juhend koondab Meshtastic sidevahendi tootmise simulatsioonist saadud teadmised praktiliseks samm-sammuliseks juhendiks, mida teised saavad järgida. Rõhutatakse avatud lähtekoodiga tarkvara, näiteks SimPy kasutamist, et minimeerida kulusid ja tagada juurdepääsetavus. See juhend on väärtuslik lähtekoht mikroettevõtetele ja iduettevõtetele, kes soovivad kasutada simulatsiooni teadlike otsuste tegemiseks ning oma tootmisprotsesside pidevaks täiustamiseks.

Kokkuvõttes pakub see magistritöö väikestele ja mikroettevõtetele uue strateegilise tööriista. Esitatud metoodika võib aidata mikroettevõtetel lihtsamalt ja väiksema finantsriskiga navigeerida tootearenduse keerukustes. Edasised uuringud võiksid käsitleda agendipõhiste simulatsioonimudelite rakendamist, turunõudluse analüüsi, toote elutsükli analüüsi ja projekti skaleeritavust.

LIST OF REFERENCES

[1] Majandus- ja Kommunikatsiooniministeerium, *Eesti tööstuspoliitika 2035*,2023. [Online]. Available: https://mkm.ee/media/9470/download.

[2] European Commission. Directorate General for Internal Market, Industry, Entrepreneurship and SMEs., *User guide to the SME definition.* LU: Publications Office,2015. [Online]. Available: https://data.europa.eu/doi/10.2873/620234 22.1.2024.

[3] Ministry of Justice. "Comparison of each form of business | eesti.ee", Riigiportaal Eesti.ee. (13.3.2024), [Online]. Available: https://www.eesti.ee/en/doing-business/establishing-a-company/comparison-of-each-form-of-business 7.4.2024.

[4] N. K. Suryadevara and A. Dutta, "Meshtastic infrastructure-less networks for reliable data transmission to augment internet of things applications", in *Wireless and Satellite Systems*, Q. Guo, W. Meng, M. Jia, and X. Wang, Eds., vol. 410, Series Title: Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Cham: Springer International Publishing,2022, pp. 622–640, ISBN: 978-3-030-93397-5. DOI: 10.1007/978-3-030-93398-2_55. [Online]. Available: https://link.springer.com/10.1007/978-3-030-93398-2_55 3.3.2024.

[5] F. Messina, C. Santoro, and F. F. Santoro, "Enhancing security and trust in internet of things through meshtastic protocol utilising low-range technology", *Electronics*, vol. 13, no. 6, p. 1055, 12.3.2024, ISSN: 2079-9292. DOI: 10.3390/electronics13061055. [Online]. Available: https://www.mdpi.com/2079-9292/13/6/1055 1.4.2024.

[6] Jonathan Bennett. "Meshtastic for the greater good", Hackaday. (26.6.2023), [Online]. Available: https://hackaday.com/2023/06/26/meshtastic-for-the-greater-good/ 14.4.2024.

[7] "Radio settings | meshtastic", [Online]. Available: https://meshtastic.org/docs/overview/radio-settings/ 1.4.2024.

[8] "Introduction | meshtastic", [Online]. Available: https://meshtastic.org/docs/introduction/ 2.4.2024.

[9] Tommy Ekstrand. "Legal | meshtastic", [Online]. Available: https://meshtastic.org/docs/legal/ 7.3.2024.

[10] Tommy Ekstrand. "Licensing & trademark rules | meshtastic", [Online]. Available: https://meshtastic.org/docs/legal/licensing-and-trademark/ 7.3.2024.

[11] "Meshtastic logo github", Meshtastic Design. (24.2.2021), [Online]. Available: https://github.com/meshtastic/design/blob/master/web/android-chrome-512x512.png 7.3.2024.

[12] Brett Smith. "A quick guide to GPLv3 - GNU project - free software foundation", GNU Operating System. (4.1.2022), [Online]. Available: https://www.gnu.org/licenses/quick-guide-gplv3.html 7.3.2024.

[13] J. Banks, J. S. Carson II, B. L. Nelson, and D. M. Nicol, *Discrete-event system simulation* (Always learning), Fifth edition, Pearson new international edition. Harlow: Pearson,2014, 559 pp., ISBN: 978-1-292-02437-0.

[14] S. Robinson, "Discrete-event simulation: From the pioneers to the present, what next?", *Journal of the Operational Research Society*, vol. 56, no. 6, pp. 619–629,6.2005, ISSN: 0160-5682, 1476-9360. DOI: 10.1057/palgrave.jors.2601864. [Online]. Available: https://www.tandfonline.com/doi/full/10.1057/palgrave.jors.2601864 2.2.2024.

[15] A. Martínez-Mingo and M. Leonelli, *1.2 Types of simulations | Simulation and Modeling to Understand the Change*. 11.3.2022. [Online]. Available: https://alejandrommingo.github.io/SMUCbook/types-of-simulations.html 3.3.2024.

[16] R. J. Madachy and D. X. Houston, *What every engineer should know about modeling and simulation* (What every engineer should know 49). Boca Raton London New York: CRC Press, Taylor & Francis Group,2018, 151 pp., ISBN: 978-1-138-29750-0.

[17] J. K. Liker, *The Toyota way: 14 management principles from the world's greatest manufacturer*, 2nd edition. New York: McGraw Hill,2021, 411 pp., ISBN: 978-1-260-46852-6.

[18] P. L. King, *Lean for the process industries: dealing with complexity*, Second edition. New York, NY: Routledge,2019, 341 pp., ISBN: 978-0-367-02332-4.

[19] S. Ahmed and M. S. I. Chowdhury, "Increase the efficiency and productivity of sewing section through low performing operators improvement by using eight wastes of lean methodology", *Global Journal of Research In Engineering*,2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:86794384.

[20] S. J. Pavnaskar, J. K. Gershenson, and A. B. Jambekar, "Classification scheme for lean manufacturing tools", *International Journal of Production Research*, vol. 41, no. 13, pp. 3075–3090,1.2003, ISSN: 0020-7543, 1366-588X. DOI: 10.1080/0020754021000049817. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/0020754021000049817 2.2.2024.

[21] D. Samuel, P. Found, and S. J. Williams, "How did the publication of the book *The Machine That Changed The World* change management thinking? exploring 25 years of lean literature", *International Journal of Operations & Production Management*, vol. 35, no. 10, pp. 1386–1407, 5.10.2015, ISSN: 0144-3577. DOI: 10.1108/IJOPM-12-2013-0555. [Online]. Available: https://www.emerald.com/insight/content/doi/10.1108/IJOPM-12-2013-0555/full/html 22.1.2024.

[22] S. A. Irani, "Advancing lean beyond the toyota way", *ISE ; Industrial and Systems Engineering at Work*, vol. 50, no. 3, pp. 42–47,2018, Place: Norcross Publisher: Institute of Industrial and Systems Engineers (IISE) tex.copyright: Copyright Institute of Industrial and Systems Engineers (IISE) Mar 2018, ISSN: 2471-9579.

[23] M. Yahya, M. Mohammad, B. Omar, and E. Ramly, "A review on the selection of lean production tools and techniques", *ARPN Journal of Engineering and Applied Sciences*, vol. 11, pp. 7721–7727,6.2016, ISSN: 1819-6608.

[24] C. Marchwinski and L. E. Institute, Eds., *Lean lexicon: a graphical glossary for lean thinkers*, Fifth edition, Cambridge, MA: Lean Enterprise Institute,2014, 131 pp., ISBN: 978-0-9667843-6-7.

[25] J. Rodrigues, J. C. Sá, F. J. G. Silva, L. P. Ferreira, G. Jimenez, and G. Santos, "A rapid improvement process through "quick-win" lean tools: A case study", *Systems*, vol. 8, no. 4, p. 55, 14.12.2020, ISSN: 2079-8954. DOI: 10.3390/systems8040055. [Online]. Available: https://www.mdpi.com/2079-8954/8/4/55 2.2.2024.

[26] N. Slack, A. Brandon-Jones, and R. Johnston, *Operations management*, Seventh edition. Boston: Pearson,2013, 733 pp., ISBN: 978-0-273-77620-8.

[27] Kent Beck, Mike Beedle, Arie van Bennekum, *et al.* "Manifesto for agile software development", Agile Manifesto. (2001), [Online]. Available: https://agilemanifesto.org/ 7.2.2024.

[28] D. Greer and Y. Hamon, "Agile software development", *Software: Practice and Experience*, vol. 41, no. 9, pp. 943–944,8.2011, ISSN: 0038-0644, 1097-024X. DOI: 10.1002/spe.1100. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/spe.1100 7.2.2024.

[29] A. H. G. Rossi, G. B. Marcondes, J. Pontes, *et al.*, "Lean tools in the context of industry 4.0: Literature review, implementation and trends", *Sustainability*, vol. 14, no. 19, p. 12 295, 27.9.2022, ISSN: 2071-1050. DOI: 10.3390/su141912295. [Online]. Available: https://www.mdpi.com/2071-1050/14/19/12295 3.3.2024.

[30] "RAK4631-8-WB-i RAKwireless technology limited | RF and wireless | DigiKey", [Online]. Available: https://www.digikey.ee/en/products/detail/rakwireless-technology-limited/RAK4631-8-WB-I/17138053 5.5.2024.

[31] "RAK19007-0-BB-n", DigiKey Electronics, [Online]. Available: https://www.digikey.ee/en/products/detail/rakwireless-technology-limited/RAK19007-0-BB-N/17138004 5.5.2024.

[32] "FW.86.b.SMA.m", DigiKey Electronics, [Online]. Available: https://www.digikey.ee/en/products/detail/taoglas-limited/FW-86-B-SMA-M/6362785 5.5.2024.

[33] "SAMSUNG laetav aku 18650 3.6v 3450mah 8a li-ion INR18650 35e SAMSUNG | LEMONA", [Online]. Available: https://www.lemona.ee/laetav-aku-18650-3-6v-3450mah-8a-li-ion-inr18650-35e-samsung.html 5.5.2024.

[34] "Patareide hoidik, juhtmega, 18650 | LEMONA", [Online]. Available: https://www.lemona.ee/patareide-hoidik-juhtmega-18650.html 5.5.2024.

[35] "851 adafruit | mouser", Mouser Electronics, [Online]. Available: https://www.mouser.ee/ProductDetail/Adafruit/851?qs=GURawfaeGuBPTe38xSkvfQ%3D%3D 5.5.2024.

[36] "2069940100", DigiKey Electronics, [Online]. Available: https://www.digikey.ee/en/products/detail/molex/2069940100/9450924 5.5.2024.

[37] "RAK1921 OLED", DigiKey Electronics, [Online]. Available: https://www.digikey.ee/en/products/detail/rakwireless-technology-limited/RAK1921-X-WB-N/22536555 5.5.2024.

[38] "EG1201a", DigiKey Electronics, [Online]. Available: https://www.digikey.ee/en/products/detail/e-switch/EG1201A/101723 5.5.2024.

[39] "261 adafruit | mouser", Mouser Electronics, [Online]. Available: https://www.mouser.ee/ProductDetail/Adafruit/261?qs=GURawfaeGuDeA9XsXeF0ug%3D%3D 5.5.2024.

[40] "Devices | supported hardware overview | meshtastic", [Online]. Available: https://meshtastic.org/docs/hardware/devices/ 2.4.2024.

[41] "RAK4631 WisBlock LPWAN module | RAKwireless documentation center", [Online]. Available: https://docs.rakwireless.com/Product-Categories/WisBlock/RAK4631/Overview/ 6.4.2024.

[42] "RAK19007 WisBlock base board 2nd gen datasheet | RAKwireless documentation center", [Online]. Available: https://docs.rakwireless.com/Product-Categories/WisBlock/RAK19007/Datasheet/#wisblock-base-board-2nd-gen-overview 6.4.2024.

[43] TonyG. "RAK19007/RAK5005 case for meshtastic", Printables.com. (28.3.2024), [Online]. Available: https://www.printables.com/model/286657-rak19007rak5005-case-for-meshtastic 6.4.2024.

[44] "T-echo meshtastic", LILYGO®, [Online]. Available: https://www.lilygo.cc/products/t-echo 22.2.2024.

[45] "Meshtastic mesh device nano g1 explorer by b&q consulting shop on tindie", Tindie, [Online]. Available: https://www.tindie.com/products/neilhao/meshtastic-mesh-device-nano-g1-explorer/ 13.4.2024.

[46] "US dollar to euro spot exchange rates for 2023", [Online]. Available: https://www.exchangerates.org.uk/USD-EUR-spot-exchange-rates-history-2023.html 14.4.2024.

[47] "Terranode complete meshtastic", Etsy, [Online]. Available: https://www.etsy.com/listing/1688511849/terranode-complete-meshtastic-device-in?utm_source=OpenGraph&utm_medium=PageTools&utm_campaign=Share 5.5.2024.

[48] S. Lang, T. Reggelin, M. Müller, and A. Nahhas, "Open-source discrete-event simulation software for applications in production and logistics: An alternative to commercial tools?", *Procedia Computer Science*, vol. 180, pp. 978–987,2021, ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2021.01.349. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050921004038.

[49] R. Van Der Ham, "Salabim: Discrete event simulation and animation in python", *Journal of Open Source Software*, vol. 3, no. 27, p. 767, 9.7.2018, ISSN: 2475-9066. DOI: 10.21105/joss.00767. [Online]. Available: http://joss.theoj.org/papers/10.21105/joss.00767 8.4.2024.

[50] "Salabim PyPI download stats", [Online]. Available: https://pypistats.org/packages/salabim 14.5.2024.

[51] *Jaamsim/jaamsim*, original-date: 2015-01-29T19:42:50Z, 31.3.2024. [Online]. Available: https://github.com/jaamsim/jaamsim 8.4.2024.

[52] Dr. Harry King and Harvey Harrison. "JaamSim- discrete-event simulation software", JaamSim- Discrete-event Simulation Software, [Online]. Available: https://www.jaamsim.com/ 8.4.2024.

[53] "OpenSIMPLY free simulation modeling software", OpenSIMPLY, [Online]. Available: https://opensimply.org/ 8.4.2024.

[54] "Team-simpy / simpy · GitLab", GitLab. (12.11.2023), [Online]. Available: https://gitlab.com/team-simpy/simpy 8.4.2024.

[55] "SimPy history & change log — SimPy 4.1.1 documentation". (13.11.2023), [Online]. Available: https://simpy.readthedocs.io/en/latest/about/history.html# 14.5.2024.

[56] "SimPy - google scholar", [Online]. Available: https://scholar.google.com/scholar?as_vis=1&q=SimPy&hl=en&as_sdt=0,5 14.5.2024.

[57] "SimPy PyPI download stats", [Online]. Available: https://pypistats.org/packages/simpy 14.5.2024.

[58] "Simpy :: Anaconda.org", [Online]. Available: https://anaconda.org/conda-forge/simpy 14.5.2024.

[59] I. Ucar, B. Smeets, and A. Azcorra, "Simmer: Discrete-event simulation for r", *Journal of Statistical Software*, vol. 90, no. 2,2019, Publisher: Foundation for Open Access Statistic, ISSN: 1548-7660. DOI: 10.18637/jss.v090.i02. [Online]. Available: http://dx.doi.org/10.18637/jss.v090.i02.

[60] *R-simmer/simmer*, original-date: 2014-08-25T11:19:09Z, 8.3.2024. [Online]. Available: https://github.com/r-simmer/simmer 14.5.2024.

[61] Ben Lauwens, *JuliaDynamics/ConcurrentSim.jl*, 1.4.2024. [Online]. Available: https://github.com/JuliaDynamics/ConcurrentSim.jl 8.4.2024.

[62] "K1 speedy 3d printer". (2024), [Online]. Available: https://www.creality3dofficial.eu/products/k1-3d-printer-eu 4.5.2024.

[63] "Creality print - creality slicer softwares download", Creality Print - Creality Slicer Softwares Download, [Online]. Available: https://www.crealitycloud.com 30.4.2024.

[64] "The main 3d printer filament types", All3DP. (12.10.2023), [Online]. Available: https://all3dp.com/1/3d-printer-filament-types-3d-printing-3d-filament/ 3.5.2024.

[65] "CR-PETG 3d printing filament 1kg", Creality Europe. (2024), [Online]. Available: https://store.creality.com/eu/products/cr-petg-3d-printing-filament-1kg 4.5.2024.

[66] A. Annamaa, "Introducing thonny, a python IDE for learning programming", in *Proceedings of the 15th koli calling conference on computing education research*, ser. Koli calling '15, Number of pages: 5 Place: Koli, Finland, New York, NY, USA: Association for Computing Machinery,2015, pp. 117–121, ISBN: 978-1-4503-4020-5. DOI: 10.1145/2828959.2828969. [Online]. Available: https://doi.org/10.1145/2828959.2828969.

[67] "Scipy.stats.truncexpon — SciPy v1.13.0 manual". (3.4.2024), [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.truncexpon.html 1.5.2024.

[68] "Numpy.random.generator.triangular — NumPy v1.26 manual". (25.1.2024), [Online]. Available: https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.triangular.html# 30.4.2024.

[69] "Numpy.random.generator.uniform — NumPy v1.26 manual". (25.1.2024), [Online]. Available: https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.uniform.html# 30.4.2024.

[70] Statistikaamet. "Harjumaa jootja keskmine palk IV kvartal 2023". (2024), [Online]. Available: https://palgad.stat.ee/# 4.5.2024.

[71] Maksu- ja Tolliamet. "Palgatulu". (2023), [Online]. Available: https://www.emta.ee/eraklient/maksud-ja-tasumine/maksustatavad-tulud/palgatulu 4.5.2024.

[72] "KV.EE: Tallinna äripindade eest küsitakse üüri keskmiselt 10,5 €/m²", [Online]. Available: https://www.kinnisvaraweb.ee/blog/kv-ee-tallinna-aripindade-eest-kusitakse-uuri-keskmiselt-105-e-m%c2%b2/ 4.5.2024.

[73] Julia Kagan. "Payback period explained, with the formula and how to calculate it", Investopedia. (23.2.2024), [Online]. Available: https://www.investopedia.com/terms/p/paybackperiod.asp 5.5.2024.

[74] Adam Hayes. "Break-even analysis: Formula and calculation", Investopedia. (2.4.2024), [Online]. Available: https://www.investopedia.com/terms/b/breakevenanalysis.asp 5.5.2024.

APPENDICES

Appendix 1: SimPy discrete event simulation code

```
import simpy
import time
import numpy as np
from scipy.stats import truncexpon

# Simulation time in hours
simulation_time_in_hours = 168

# Number of printer
NUM_PRINTERS = 7

# Debug output flag
debug_output = True

# Set up the random generator with a fixed seed
seed = 2024  # Makes simulation reproducible
rng = np.random.default_rng(seed)

# Constants for flash duration
MIN_FLASH = 290 # seconds
MODE_FLASH = 310 # seconds
MAX_FLASH = 370 # seconds

# Constants for assembly duration
DELTA_ASSEMBLY = 30 # seconds
BASE_ASSEMBLY = 490 # seconds

# Constants for soldering duration
DELTA_SOLDERING = 45 # seconds
BASE_SOLDERING = 653 # seconds

# Constants for print duration
BASE_PRINT = 8891 # seconds
MAX_PRINT = 2 * BASE_PRINT
SCALE_PRINT = 100
B_PRINT = (MAX_PRINT - BASE_PRINT) / SCALE_PRINT
PRINTER_RESET = 60 # seconds


# Function definitions
def flash_duration(): # Generate flash duration using triangular distribution
    global MIN_FLASH, MODE_FLASH, MAX_FLASH
```

```
    return rng.triangular(MIN_FLASH, MODE_FLASH, MAX_FLASH)

def assembly_duration(): # Generate assembly duration using uniform distribution
    global BASE_ASSEMBLY, DELTA_ASSEMBLY
    return rng.uniform(BASE_ASSEMBLY - DELTA_ASSEMBLY, BASE_ASSEMBLY + DELTA_ASSEMBLY)

def soldering_duration(): # Generate soldering duration using uniform distribution
    global BASE_SOLDERING, DELTA_SOLDERING
    return rng.uniform(BASE_SOLDERING - DELTA_SOLDERING, BASE_SOLDERING + DELTA_SOLDERING)

def print_time(): # Generate print time using truncated exponential distribution
    global B_PRINT, SCALE_PRINT
    return BASE_PRINT + truncexpon.rvs(B_PRINT, scale=SCALE_PRINT)


# Global counters
total_parts_printed = 0
total_pcbs_flashed = 0
total_pcbs_soldered = 0
products_assembled = 0
wip_parts = [0] * NUM_PRINTERS
wip_pcbs_flashed = 0
wip_pcbs_soldered = 0

# Utilization tracking variables
worker_busy_time = 0
printer_busy_times = [0] * NUM_PRINTERS
flashing_busy_time = 0
soldering_busy_time = 0
assembly_busy_time = 0

def printer(env, printer_id, start_delay):
    global total_parts_printed, printer_busy_times
    yield env.timeout(start_delay)
    while True:
        if wip_parts[printer_id] < 1:
            start_time = env.now
            if debug_output:
                print(f"Printer {printer_id} started at {start_time:.2f}")
            yield env.timeout(print_time())
            printer_busy_times[printer_id] += env.now - start_time
            total_parts_printed += 1
            wip_parts[printer_id] += 1
            if debug_output:
                print(f"Printer {printer_id} finished at {env.now:.2f}. Parts waiting: "
                      f"{wip_parts[printer_id]}")
        else:
            yield env.timeout(1)

def operator_workflow(env):
    global products_assembled, total_pcbs_flashed, total_pcbs_soldered
    global wip_pcbs_flashed, wip_pcbs_soldered, worker_busy_time
    global flashing_busy_time, soldering_busy_time, assembly_busy_time
```

```
    while True:
        # Flashing process
        start_time = env.now
        yield env.timeout(flash_duration())
        duration = env.now - start_time
        worker_busy_time += duration
        flashing_busy_time += duration

        total_pcbs_flashed += 1  # PCB: Printed Circuit Board
        wip_pcbs_flashed += 1
        if debug_output:
            print(f"PCB flashed at time {env.now}. PCBs waiting for soldering: "
                  f"{wip_pcbs_flashed}")

        # Soldering process
        if wip_pcbs_flashed > 0:
            wip_pcbs_flashed -= 1
            start_time = env.now
            yield env.timeout(soldering_duration())
            duration = env.now - start_time
            worker_busy_time += duration
            soldering_busy_time += duration

            total_pcbs_soldered += 1
            wip_pcbs_soldered += 1
            if debug_output:
                print(f"PCB soldered at time {env.now}. PCBs waiting for assembly: "
                      f"{wip_pcbs_soldered}")

        # Collect parts from printers
        while sum(wip_parts) == 0:
            yield env.timeout(1)

        for printer_id in range(NUM_PRINTERS):
            if wip_parts[printer_id] > 0:
                start_time = env.now
                yield env.timeout(PRINTER_RESET)
                duration = env.now - start_time
                worker_busy_time += duration
                wip_parts[printer_id] -= 1
                if debug_output:
                    print(f"Collected part from Printer {printer_id} at time {env.now}.")
                break

        # Assembly process
        if wip_pcbs_soldered > 0:
            wip_pcbs_soldered -= 1
            start_time = env.now
            yield env.timeout(assembly_duration())
            duration = env.now - start_time
            worker_busy_time += duration
            assembly_busy_time += duration
            products_assembled += 1
```

```
            if debug_output:
                print(f"Product assembled at time {env.now}. Total assembled: "
                      f"{products_assembled}")

def calculate_start_delay(printer_id):
    return printer_id * PRINTER_RESET

# Set up the simulation environment
env = simpy.Environment()
for i in range(NUM_PRINTERS):
    env.process(printer(env, i, calculate_start_delay(i)))
env.process(operator_workflow(env))

# Run the simulation
simulation_time = simulation_time_in_hours * 60 * 60
start_time = time.time()  # Record the start time of the simulation

env.run(until=simulation_time)

end_time = time.time()  # Record the end time of the simulation
elapsed_time = end_time - start_time  # Calculate how long the simulation took

# Print simulation results
print(f"Total Parts Printed: {total_parts_printed}")
print(f"Total PCBs Flashed: {total_pcbs_flashed}")
print(f"Total PCBs Soldered: {total_pcbs_soldered}")
print(f"Total Products Assembled: {products_assembled}")

assembly_utilization = (assembly_busy_time / simulation_time) * 100
flashing_utilization = (flashing_busy_time / simulation_time) * 100
soldering_utilization = (soldering_busy_time / simulation_time) * 100
worker_utilization = (worker_busy_time / simulation_time) * 100

print(f"Assembly Station Utilization: {assembly_utilization:.2f}%")
print(f"Flashing Station Utilization: {flashing_utilization:.2f}%")
print(f"Soldering Station Utilization: {soldering_utilization:.2f}%")
print(f"Worker Utilization: {worker_utilization:.2f}%")

total_printer_busy_time = sum(printer_busy_times)
time_per_printer = simulation_time * NUM_PRINTERS
utilization_ratio = total_printer_busy_time / time_per_printer
total_printers_utilization = utilization_ratio * 100

for i, time in enumerate(printer_busy_times):
    printer_utilization = (time / simulation_time) * 100
    print(f"Printer {i} Utilization: {printer_utilization:.2f}%")

print(f"Total Printers Utilization: {total_printers_utilization:.2f}%")
print(f"Simulation took {elapsed_time:.2f} seconds.")

average_assembly_time = simulation_time / products_assembled if products_assembled else 0
print(f"Average Assembly Time per Product: {average_assembly_time:.2f} seconds")
```

Appendix 2: Guide for implementing affordable simulation

Guide for implementing affordable simulation

This checklist provides guidance on how to implement affordable simulation techniques. The following steps and provided comments may help you optimize your manufacturing processes and make informed decisions.

1. **Determine if simulation is necessary:**
    - What are your primary goals for analyzing your manufacturing process?
    - How complex is your production line and can it be easily understood without simulation?
    - Will simple calculations or napkin math suffice for your analysis needs?

    Comment: Simulation is a powerful tool, but it may not be necessary for every situation. If your manufacturing process is relatively straightforward and can be analyzed using basic math, you might be able to make informed decisions without investing time and resources into simulation. However, if your process involves complex interactions, bottlenecks, or variability, simulation can provide valuable insights.

2. **Identify the appropriate type of simulation:**
    - What level of detail do you need to capture in your simulation?
    - Are you primarily interested in modeling discrete events or continuous flows?
    - Do you need to account for the individual behaviors and interactions of autonomous agents?

    Comment: Discrete event simulation (DES) is the most common choice for manufacturing simulations, as it focuses on modeling discrete events and processes. DES is well-suited for analyzing queues, resource utilization, and bottlenecks. If you need to model continuous flows or feedback loops, system dynamics simulation might be more appropriate. Agent-based modeling is useful when you need to account for individual behaviors and interactions of autonomous agents, such as in supply chain simulations.

3. **Select a suitable open-source discrete event simulation software:**

    - What programming languages are you comfortable with?
    - Do you require a graphical user interface, or are you comfortable with coding?
    - What level of community support and documentation do you need?

    Comment: If you have a preferred programming language, try searching for libraries for that language. If you don't have a preferred language, then look at Python, since it's relatively easy to learn and has a wide variety of packages available for it. If you have settled on using DES, consider SimPy, a popular Python-based library. SimPy is widely used, easy to learn, and has strong community support, making it an excellent choice for beginners. Additionally, Thonny is a beginner-friendly integrated development environment (IDE) that can help you get started with Python.

4. **Design your manufacturing layout and estimate process times:**

    - Map out your production line and identify its key components.
    - Break down your production processes into small, manageable steps.
    - Estimate the time required for each step based on your resources and tools.

    Comment: Designing an accurate representation of your manufacturing layout is crucial for developing a useful simulation model. Design of the production line and breaking down of the production process is invaluable input for the simulation. Be careful not to break down the process into too large or too small steps. as that will hinder the time estimations. Consider applying lean principles and cellular manufacturing concepts to optimize your layout for efficiency and scalability if you are at the early stages of process development.

5. **Develop the simulation model:**

    - Identify the key elements of your chosen simulation approach (e.g., entities, resources, activities, sources, sinks).
    - Map these elements to your process and create a conceptual model.
    - Implement your conceptual model using the selected simulation software.
    - Start with a minimal viable simulation and gradually add more detail as needed.
    - Verify the simulation logic by testing with different input parameters and analyzing outputs.

Comment: Developing a simulation model involves translating your manufacturing process into the language of your chosen simulation approach. For DES, this typically means identifying entities (products), resources (machines, workers), and activities (process steps). Map these elements to your production line and create a conceptual model. Then, implement this conceptual model using your selected simulation software. Start with a minimal viable simulation and verify its logic by testing with different input parameters. Gradually add more detail as needed, but remember that the goal is not perfect realism but rather a useful approximation.

6. **Analyze simulation output and optimize manufacturing setup:**

   - Identify the key performance indicators (KPIs) relevant to your goals.
   - Run multiple simulation scenarios with different input parameters.
   - Analyze the simulation output to identify bottlenecks, resource utilization, and improvement opportunities.
   - Use the insights gained from the simulation to optimize your manufacturing setup or to determine whether manufacturing this product is even viable.

Comment: The ultimate goal of simulation is to provide actionable insights for optimizing your manufacturing process. Start by identifying the KPIs that are most relevant to your goals, such as throughput, cycle time, or resource utilization. Run multiple simulation scenarios with different input parameters to explore the impact of various factors on your KPIs. Use sufficiently long simulation durations to account for the buildup of work-in-progress and capture steady-state behavior. Analyze the simulation output to identify bottlenecks, resource utilization patterns, and potential areas for improvement. Use these insights to make informed decisions about optimizing your manufacturing setup, such as adjusting resource allocation, modifying process steps, or implementing lean principles.