Resilient by Design: A Supply Chain Digitalization Journey

by

Carlos David Vela González

B.S. in Industrial Engineering with minor in Systems Engineering
Tecnológico de Monterrey, 2017

Submitted to the MIT Sloan School of Management and
Department of Civil and Environmental Engineering
in partial fulfillment of the requirements for the degrees of

Master of Business Administration

and

Master of Science in Civil and Environmental Engineering

in conjunction with the Leaders for Global Operations program

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2024

Authored by . . . .
MIT Sloan School of Management and
Department of Civil and Environmental Engineering
May 10, 2024

Certified by . . . .
Dr. Thomas Roemer, Thesis Supervisor
Senior Lecturer in Operations Management

Certified by . . . .
Dr. David Simchi-Levi, Thesis Supervisor
Professor of Civil and Environmental Engineering

Accepted by . . . .
Heidi Nepf
Donald and Martha Harleman Professor of Civil and Environmental Engineering;
MacVicar Faculty Fellow; Graduate Officer

Accepted by . . . .
Maura Herson
Assistant Dean, MBA Program MIT Sloan School of Management

Resilient by Design: A Supply Chain Digitalization Journey

by

Carlos David Vela González

Submitted to the MIT Sloan School of Management and
Department of Civil and Environmental Engineering
on May 10, 2024, in partial fulfillment of the
requirements for the degrees of
Master of Business Administration
and
Master of Science in Civil and Environmental Engineering

Abstract

In an era where supply chain disruptions have become increasingly relevant due to geopolitical and environmental factors, resilience has emerged as a critical focus for organizations worldwide. This is particularly true in the pharmaceutical sector, where ensuring an uninterrupted supply of medical products is not only a business necessity but also a moral imperative, given the direct impact on patients' health and well-being.

This thesis presents the development of a digital tool designed to enhance the resilience of AstraZeneca's supply chain, employing a design thinking approach. The tool leverages simulation and business intelligence, providing a versatile platform for conducting stress tests and evaluating response mechanisms across a spectrum of scenarios. This capability is instrumental in refining business continuity plans and informing strategic decisions on disruption response and capacity investments.

While the tool was initially conceived to address the specific needs of AstraZeneca, its architecture is inherently generic and modular. This deliberate design choice ensures that the tool can be seamlessly adapted and scaled for use across various industries, transcending the initial scope of application. Additionally, the tool lays a solid foundation for future developments in the realm of supply chain digital twins.

The thesis also contributes a comprehensive framework for boosting supply chain resilience through the lens of digitalization. It offers a strategic blueprint that organizations can adopt to proactively navigate and mitigate the intricacies of global supply chain disruptions.

Thesis Supervisor: Dr. Thomas Roemer
Title: Senior Lecturer in Operations Management

Thesis Supervisor: Dr. David Simchi-Levi
Title: Professor of Civil and Environmental Engineering

Acknowledgments

I would like to extend my heartfelt gratitude to the LGO program for giving me this once-in-a-lifetime opportunity and to AstraZeneca for hosting my internship and entrusting me with this project.

Special thanks to Thomas Wheatley, my supervisor, for playing a decisive role in the success of my project, for his remarkable mentorship and friendship, and for immersing me in British culture. Many thanks to Jo Kirby and the ESM team for sponsoring my project, to Bastien Hermes for his unparalleled support and camaraderie, to Ian Clegg and the Steering Committee for their unwavering guidance, and to Magnus Torstensson, his team, and AZDP's suppliers, for facilitating my visits to their manufacturing sites and for providing invaluable insights into the brand supply chain.

I am also grateful to Thomas Roemer and David Simchi-Levi, my academic advisors, for their commitment to my project. Their time, expertise, and constructive suggestions were instrumental in shaping the trajectory of my research.

I want to express my deepest appreciation to Lari, my dear wife. Thank you for filling my heart with happiness every day, for embarking with me on this journey, and for demonstrating your unconditional love and support at all times. Thank you for all these wonderful years and those to come - for always glowing in the dark.

To my mom, thank you for pushing me to strive for excellence in all that I do and for being my source of inspiration. I am eternally grateful for everything you do for me and my sisters, for your absolute love and devotion, for being your son.

To my sisters, thank you for always being there, for being a pillar of love in my life, and for sharing infinite laughs - I am truly blessed to have you. To my GOM family, thank you for showing me the utmost meaning of family - you hold a special place in my heart and I am forever indebted to you. To my father, thank you for teaching me valuable life lessons. To those who are no longer here, thank you for shaping my life. And to my in-laws, thank you for welcoming me into your family from day one.

Finally, I wish to thank everyone, from friends to faculty, at MIT, ITESM, and CVH who has contributed to my academic, professional, and personal development.

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

Acronyms

AI artificial intelligence. 31, 36, 85

API active pharmaceutical ingredient. 26–28

AZ AstraZeneca. 5, 17–19, 21, 25–28, 31, 32, 39–41, 45, 46, 66, 70, 71, 81, 82, 87, 89

AZDP AstraZeneca Dunkerque Production. 5, 27, 43, 69, 71

BCP business continuity plan. 20, 21, 25, 39–42, 46–48, 51, 65, 66, 68, 70, 77, 81, 87

BOM bill of materials. 26, 45, 46, 63, 64

DT digital twin. 9, 36–38, 41, 42, 81, 85, 86, 88

ESM External Supply and Manufacturing. 5, 19, 21, 40, 41, 66, 67, 71

GSC&S Global Supply Chain and Strategy. 19, 21, 40, 66

OEE overall equipment effectiveness. 41, 45, 52, 55, 57, 63, 65

pMDI pressurised metered-dose inhaler. 9, 25, 26

POC proof of concept. 21, 32, 41, 42, 48

PT&D Pharmaceutical Technology and Development. 19

R&D research and development. 17, 31

TAFD time-away-from-design. 43, 46, 75

TTR time-to-recover. 34, 35, 43, 45, 65, 72, 75

TTS time-to-survive. 35, 43, 45, 46, 73, 82

VSM value stream mapping. 41

WIP work in progress. 45, 55–57, 61, 64, 72

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 1

Introduction

1.1 Company Overview

AstraZeneca (AZ) is a pharmaceutical company that serves over 116 million patients across 125 countries, focusing on three major therapy areas: oncology, biopharmaceuticals[1], and rare diseases [1].

The company follows a business model that revolves around the entire medicine's life-cycle, shown in Figure 1-1. It starts by investing in research and development (R&D) to discover drugs that can potentially treat diseases. Promising discoveries then go through a series of trials to prove the safety and effectiveness of the drug in scope, after which the company requests regulatory approval. Once a medicine gains authorization, AZ typically has a period of manufacturing exclusivity during which the company can recover most of the investment through their high-profit margin, over 80% according to the company's full-year 2023 results [2]. A significant portion of the sales revenue is later reinvested into R&D to help fund discoveries, kick-starting a new cycle. This business model translates into a virtuous cycle of innovation that enables AZ to fulfill their purpose "[to] push the boundaries of science to deliver life-changing medicines" and stay true to one of their key values "put patients first" [3].

On top of AZ's existing broad portfolio of medicines, they have a robust pipeline

[1] Encompasses three therapy areas: cardiovascular, renal and metabolism; respiratory and immunology; vaccines and immune therapies

of 178 potential new medicines that are undergoing clinical trials, launched three new medicines in 2023, and, according to their CEO Pascal Soriot, are on track to launch at least 12 more by 2030 [1].

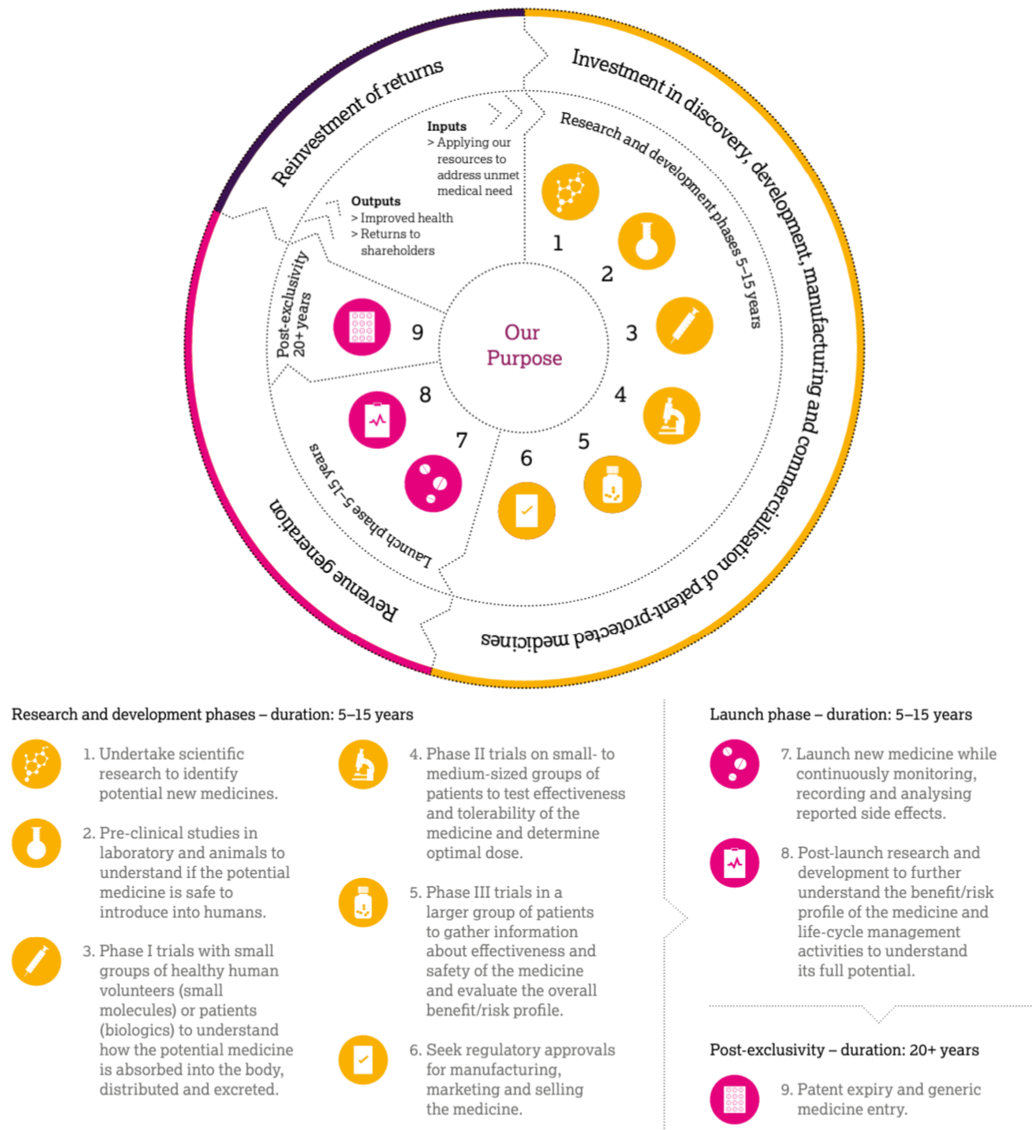


Figure 1-1: Description of the high-level overview of a medicine's life-cycle [1]

To mass-produce their existing brands, as of December 2023, AZ had 27 manufacturing sites spread across 16 countries, as shown in Figure 1-2, but those sites do not cover the entire value of production; therefore, the company also has a network of 57,000 suppliers worldwide [1][4]. To ensure impeccable execution throughout the entire supply chain and on-time delivery to the patients, three main functions collaborate within the broader AZ Global Operations department: (1) External Supply and Manufacturing (ESM), responsible for handling the relationship with suppliers and accountable for the suppliers' performance; (2) Global Supply Chain and Strategy (GSC&S), responsible for the brands' P&L and end-to-end supply chains; and (3) Pharmaceutical Technology and Development (PT&D), responsible for the technical development of the products.

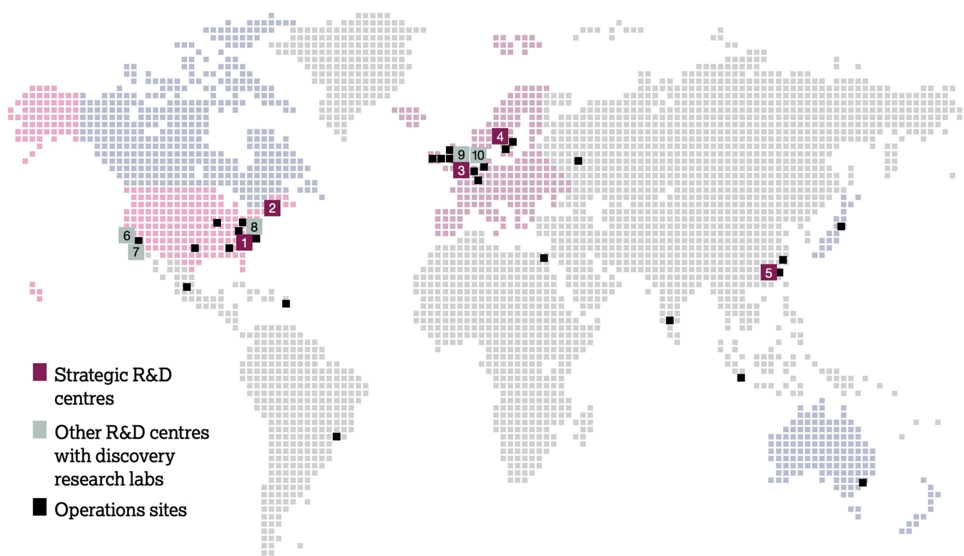


Figure 1-2: AstraZeneca Global Presence [1]

Note: adapted figure from the original source to include footnotes

1.2 Project Motivation

AZ, and pharmaceutical companies in general, have the inherent responsibility to have an impeccable service level because, most times, patients' lives depend on it. Meeting this standard is very challenging because unpredictable events constantly threaten to disrupt supply chains for prolonged periods.

A simplistic way to address this concern, and possibly the only solution that guarantees no stockouts, would be maintaining infinite amounts of inventory for each finished good that a pharmaceutical company manufactures, but this idea is economically infeasible and physically impossible. Instead, companies focus on reducing the stockout risk by understanding the potential impact associated with various disruptions and defining a business continuity plan (BCP) for their supply chains.

Implementing a BCP is a good management practice to improve the resilience of a supply chain; however, outlining the BCP is not trivial because of three main reasons:

1. *Supply chains are dynamic.* The number of nodes - entities - in the supply chain and the location of such nodes change over time. For example: a company may need to add more suppliers to meet the increasing demand for a product, change suppliers after a contract expires, or even cut suppliers if they decide to vertically integrate. *Outlining a BCP requires recurrent revisions to reevaluate risks.*

2. *Decisions are path-dependent.* Actions taken today may limit what can be done in the future. For example, if a company decides to expand the warehouse at a manufacturing site, that may cause spatial constraints to install more production lines in the future or financial constraints for several months before being able to invest in other backup mechanisms throughout the supply chain. *Outlining a BCP requires pondering present and future risks.*

3. *Visibility across supply chain tiers is limited.* The further away a supply chain node is from a company, the more difficult it becomes to access information about such a node. For example: if a company wants to know what the capacity of a tier-two supplier is, they first need to approach the tier-one supplier to know who their tier-two suppliers are and ask for a contact. *Outlining a BCP requires collaborating with external stakeholders.*

Moreover, while defining and before implementing a BCP, testing the effectiveness of the prevention and reaction strategies included in a BCP is crucial. Yet, in most supply chains, understanding the ripple effect of one or multiple disruptions is not straightforward. Consequently, defining BCPs usually becomes a sophisticated decision-making exercise that takes several days, involves large teams, and barely convinces stakeholders.

1.3 Problem Statement

Managing supply chains at AZ is an ongoing challenge that balances two responsibilities: delivering medicines to patients at the right time and contributing towards a positive P&L statement. The expanding portfolio of medicines, the large global network of suppliers, the component interdependencies across brands, the intellectual property of hardware and software, and the pharmaceutical regulatory compliance exacerbate the complexity for supply chain managers of the ESM and GSC&S teams.

The objective of the underlying project of this thesis was to develop an in-house tool that could mainly:

1. Allow AZ supply chain managers to test a product's end-to-end supply chain under different stress scenarios for BCP purposes

2. Serve as the proof of concept (POC) for a supply chain digital twin

1.4 Thesis Overview

This thesis presents a comprehensive exploration of the development and application of a digital tool designed to enhance supply chain resilience within AstraZeneca (AZ). The following chapters detail the journey from conceptualization to realization, reflecting on the broader implications of digital innovation in supply chain management.

Chapter 2 introduces AZ's brand portfolio that motivated this thesis project, providing a high-level overview of the products and the supply chain in scope. It also provides context about why supply chain resilience has become relevant and why any supply chain stakeholder should care about resilience. Finally, it gives background on how this project fits within the company given their ongoing digitalization strategy.

Chapter 3 studies the ideal features and basic considerations that the thesis project tool should encompass to be effective and solve the problem in scope. First, it examines the concept of supply chain resilience, identifies two major strategies for embracing supply chain resilience, recognizes the difficulty of convincing stakeholders to adopt resilience, and pinpoints two widely adopted metrics to expose supply chain risks. Then, it develops a better understanding of the digital twin by dissecting its characteristics and clarifying misconceptions.

Chapter 4 provides a comprehensive narrative of the tool's development journey following a design thinking approach. It explains how each stage of the methodology builds upon the previous, showcasing an iterative process that culminates in a functional tool aimed at enhancing supply chain resilience practices. It is divided into five stages, from understanding user needs to testing its efficacy in a real-world business scenario.

- *Chapter 4.2* introduces the Empathize Stage, a foundational phase where stakeholder interviews were conducted to gather insights and establish the project's baseline requirements. It details the main considerations accounted for while developing the tool.

- *Chapter 4.3* introduces the Define Stage, a phase focused on narrowing down the problem and setting clear objectives for the tool. It details the key decisions made regarding the types of disruptions to assess, performance metrics, and the scope of the supply chain to be included in the tool.

- *Chapter 4.4* introduces the Ideate Stage, a phase for designing the tool's logical framework and conceptualizing a model for simulating the supply chain logic under various disruptions. It explains some of the main assumptions and boundaries set to ensure the model's manageability and applicability, focusing on the essential elements that would allow for effective stress testing and BCP development.

- *Chapter 4.5* introduces the Prototype Stage, a phase centered on translating the conceptual model into a working simulation tool. It details the coding process, the creation of a user-friendly interface, and the development of a visual dashboard for analyzing the supply chain.

- *Chapter 4.6* introduces the Test Stage, a phase serving as a practical demonstration of the tool's capabilities within a real business context. It bridges the gap between the theoretical development in the earlier stages and the tool's practical application, aligning with the project's goals of enhancing supply chain resilience practices.

Chapter 5 recapitulates the methodical development of the tool and outlines a framework for organizations looking to bolster supply chain resilience through digital innovation. It also discusses the evolution of digital models in supply chain management towards digital twins and touches on the strategic considerations organizations must make when developing supply chain digital twins.

Chapter 6 concludes by highlighting the creation of a digital tool that enhances supply chain resilience and underscores the importance of collaboration and the potential for broader adoption of such digital tools in supply chain management.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 2

Background

2.1 Product Brand Overview

In 2016, the FDA approved 'Device A1' (name redacted for this thesis), AZ's first pressurised metered-dose inhaler (pMDI) that delivers the drug using a new patented technology - referred to as 'pmdiTech' for this thesis [5]. Up to date, two more products based on the same delivery technology have been FDA approved; 'Device A2' (name redacted for this thesis) in 2020 [6] and 'Device A3' (name redacted for this thesis) in early 2023 [7]. This family of products is used to treat respiratory conditions such as asthma or chronic obstructive pulmonary disease.

The pmdiTech portfolio of devices was selected as the brand in scope for this thesis project because it exemplifies the complexity of performing BCPs. The products' multinational presence is vast and still growing (see Table A.1), each product is configured differently (see Figure A-1 and Table A.3), and the supply chains are extensive.

The case of Device A2, the most relevant product of the portfolio in terms of sales (see Table A.2), makes the brand further interesting from a supply chain management perspective. First, the aggressive growth it has seen so far may be boosted if approved for additional purposes that are currently undergoing Phase III trials [8][9], adding uncertainty to the portfolio's demand forecast and incentivizing the company to invest in a new manufacturing site in China [10]. Second, the device is completing a transition to a newer design (see Figure A-2). Third, it is the pmdiTech product with the most complex bill of materials (BOM). Fourth, the product is expected to experience further changes soon as part of AZ sustainability initiatives [11].

2.1.1 Devices

pMDIs have existed since the 1950s and have certainly evolved; nevertheless, the general illustration provided by Newman (2006), shown in Figure 2-1, remains valid. Newman describes a pMDI as "a small portable device ... ready for use" consisting of "an aluminum can mounted in a plastic actuator" that delivers doses of a drug "as a spray via a sophisticated metering valve" and such drug "is usually a micronized suspension of drug particles but may be a solution dissolved in propellants, ethanol, or another excipient" [12].

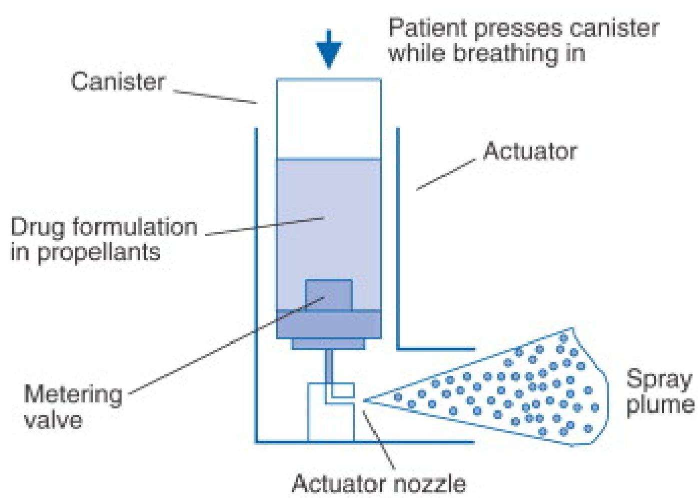


Figure 2-1: Simplified design of a pMDI [12]

To obtain nearly all the components that make up the inhaler, AZ relies on external suppliers and groups them into five different categories:

- *Starting Materials* - chemical substances required to produce the active pharmaceutical ingredient (API)
- *API and Excipients*

- *Formulation* - propellants
- *Devices* - canister, valve, and plastic components
- *Assembly and Packing* - aluminum foil pouch and desiccant

Although all suppliers are critical for manufacturing the inhalers, the current design of pmdiTech products (see Figure A-3) suggests there may be further risks within the valve supplier and the dose counter supplier. Both are sub-assembled components with various smaller pieces and, potentially, more complex manufacturing processes.

2.1.2 Supply Chain

The commercial production of the pmdiTech inhalers starts at some of the multiple AZ facilities around the world that specialize in making APIs using starting materials. Once produced, the APIs are shipped to the AstraZeneca Dunkerque Production (AZDP) manufacturing site in France - the main pmdiTech facility for the scope of this thesis - specialized in putting together the rest of the inhaler components.

AZDP has filling, assembly, and packing lines:

- Filling lines perform three activities: formulate the drug by mixing the APIs with some excipients and propellants, seal the aluminum canisters with a valve, and fill the cans with the drug through the valves.
- Assembly lines place the filled cans inside the plastic actuator and attach the plastic dose counter to the bottom side of the can.
- Packing lines place the assembled inhaler inside an aluminum foil pouch alongside a desiccant. Then, the sealed pouch goes inside a cardboard box alongside an Instructions for Use manual.

Finally, the finished goods are shipped and stored in distribution centers at the corresponding destination country to fulfill the local demand.

The production process described above applies to most product-customer combinations, but local pharmaceutical regulations in the destination country may constrain the real supply chains. For example, to comply with Japanese regulations, the pmdiTech inhalers must be packed in Japan; therefore, at the end of the assembly lines at the main pmdiTech facility, some inhalers are segregated and shipped to Japan, where another AZ site takes care of the packing process.

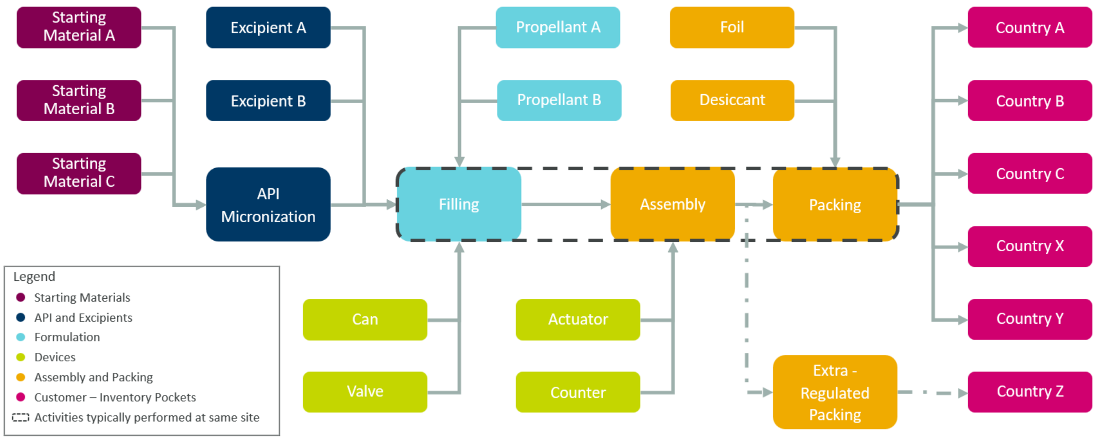


Figure 2-2: Simplified representation of an pmdiTech supply chain

A sample supply chain for the pmdiTech portfolio can be seen in Figure 2-2. Such representation contemplates two caveats. First, the number of boxes in the diagram is not necessarily the number of nodes that the real supply chains have. Second, the number of components shown in the diagram, as well as the quantity of starting materials, APIs, excipients, propellants, and destination countries may vary depending on the product.

Moreover, AZ already implements a multi-sourcing strategy for most components to diversify operational risks, meet the growing demand, and comply with the ongoing product changes. Multi-sourcing strategies prove effective in increasing supply chain resilience as long as no upstream interdependencies exist between the suppliers providing the same component. The intricacy of these interactions is not always intuitive and it modifies how ripple effects propagate through the supply chain when disruptions occur, which translates into an opportunity to develop visualization solutions for supply chain management.

2.2 Supply Chain Disruptions Overview

Supply chains have always been exposed to disruptions of greater or lesser scale, but recent global events have revealed the critical need for supply chain resilience.

The COVID-19 pandemic served as a large-scale experiment to demonstrate the disproportionate effects that a prolonged disruption can have on the financial health of any industry that prioritizes cost minimization without concern for building resilience into its supply chains; it served to confirm the proverb that a supply chain is as fragile as its weakest link. Sanitary measures imposed by governments prevented the production of non-essential goods for weeks or even months, depending on the state and country, leading to shortages and consequently higher prices that later translated into inflation.

Figure 2-3 illustrates the correlation between inflation and supply chain disruptions by overlaying the year-over-year change in the Consumer Price Index and the Global Supply Chain Pressure Index, an indicator developed by the Federal Reserve of New York over two years after the pandemic started [13].

Pre-pandemic, both indices followed a lateral trend, but the public health emergency generated a new dynamic from 2020 onwards. The Global Supply Chain Pressure Index jumped immediately and, although the inflation remained muted for a few months given the recessive economic activity, towards 2021 the Consumer Price Index started mirroring the supply chain index behavior with a lag. New COVID-19 strains and frequent lockdowns interrupted global commerce through labor and material shortages, stressing both indices during 2021 and 2022, but those pressures eventually eased as vaccinations were administered. By early 2023, the Global Supply Chain Pressure Index returned to average levels, whereas the year-over-year change in the Consumer Price Index reached zero almost a semester later.

The pandemic revealed, through inflation, a severe consequence of supply chain disruptions that in hindsight may seem obvious, but was not. The fact that government agencies, such as The White House, became concerned about supply chain resilience speaks volumes about its relevance [14].

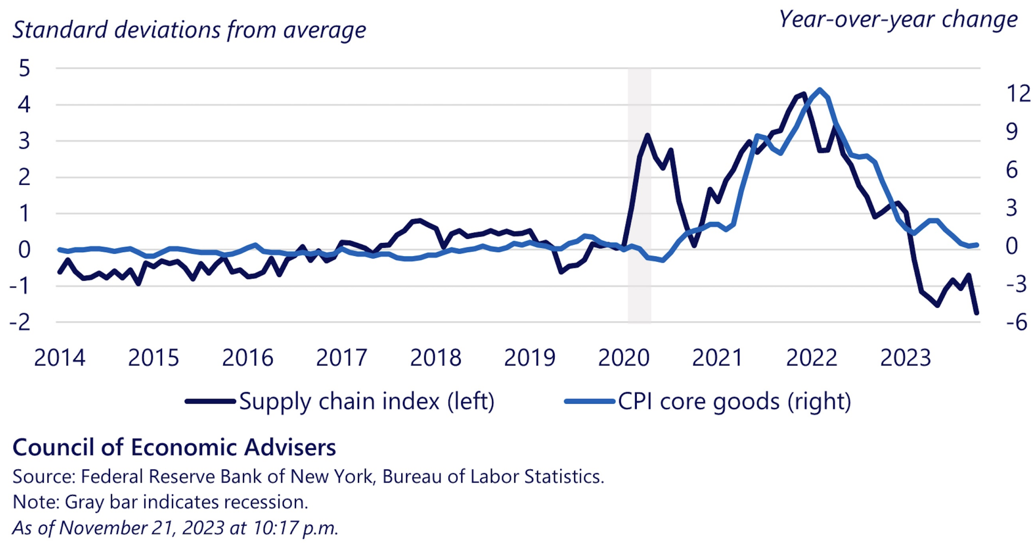


Figure 2-3: Supply chain disruptions effect on inflation [14]

Aside from the global pandemic, other recent disruptions of various types have also had a major impact:

- On the geopolitical front, two ongoing armed conflicts, one between Russia and Ukraine and the other in the Gaza Strip, are displacing supply chains operating in those areas. Tension has also escalated between China and Taiwan, a relationship in which the United States is involved, and the mere potential for conflict is prompting the semiconductor industry to diversify; TSMC, an Apple supplier, is building factories outside Taiwan [15], while the U.S. government enacted the CHIPS and Science Act to encourage the manufacture of these semiconductors on U.S. soil [16].

- On the environmental front, strong earthquakes and hurricanes, such as those in Turkey-Syria or Acapulco, Mexico in 2023 respectively, are usually the most devastating natural disasters. However, global warming is causing wildfires, floods, landslides, and winter storms to occur more frequently and become more relevant in establishing logistical routes [17].

- On the operational front, a ship ran aground in the Suez Canal blocking for six days, in March 2021, the route through which 12% of global trade travels [18].

Human error or machine failure can also affect other nodes in the supply chain.

- On occasions, multiple types of disruptions overlap. Recently, the Panama Canal has suffered from droughts, while terrorist attacks in the Red Sea have discouraged traffic through the Suez Canal [19]. Beyond affecting delivery times and prices, these two events exemplify that disruptions may be uncorrelated but simultaneous.

The combined effect of all the disruptions mentioned above may feel unprecedented but created a much-needed sense of awareness about the importance of resilience in supply chains.

A McKinsey survey of supply chain leaders in May 2020 showed that 93% of the respondents indeed "plan(ned) to increase resilience across the supply chain" [20]. Reponses from subsequent surveys show that the most common actions were to increase inventories and implement dual-sourcing strategies, but the most effective actions revolved around gaining end-to-end visibility, scenario planning, and having good-quality data [21].

The unpredictable nature of disruptions only suggests that supply chain resilience is here to stay. The goal is to find ways to incorporate resilience smoothly into the everyday lives of supply chain managers and identify opportunities to build competitive advantages throughout the process.

2.3 Digital Landscape Overview

When thinking about AZ's business model, one inevitably thinks about R&D at the core. Naturally, most of the technological advancements are applied toward creating novel medicines more efficiently. For instance, they are using artificial intelligence (AI) to analyze pathologies, grow their drug pipeline, and design better clinical trials; they are developing new therapeutic modalities; and they even created a dedicated health-tech business unit to continue improving their products and services [22] [23].

To keep up to speed with their growing pipeline, in 2021 AZ decided to undertake an Operations 2025 plan that focused on "leveraging the benefits of new manufacturing technology and digital innovation" among other goals [24].

During the first phase of the plan, AZ identified seven priority areas - ‘building blocks’ - where the company should focus its technological development efforts. More importantly, the company defined sites - ‘digital lighthouses’ - that will verify if the proof of concept (POC) of such technological initiatives drives business value before scaling the technology and implementing it across the organization [25] [26].

As a follow-up to their Operations 2025 plan, in 2023 AZ announced Operations 2030. This new stage of their supply chain digitalization journey will focus on "implementing next-generation manufacturing technologies and smart factory capabilities" [27]. This imminent supply chain transformation suggests that right now is an appropriate moment to try to introduce resilience into the agenda.

Chapter 3

Literature Review

3.1 Supply Chain Resilience

Pomonarov and Holcomb [28] comprehensively defined supply chain resilience as *"The adaptive capability of the supply chain to prepare for unexpected events, respond to disruptions, and recover from them by maintaining continuity of operations at the desired level of connectedness and control over structure and function"*.

Sheffi [29] identified two avenues to improve supply chain resilience: redundancy and flexibility - with the latter offering a competitive edge. Redundancy usually comes as excess inventory or capacity, such as dual-sourcing strategies or having backup tools, machines, or sites. Flexibility has more to do with designing or reconditioning products and processes to serve more than one purpose; an analogous example would be training employees to operate any machine at a site in case of unplanned staff shortages in one area.

Both supply chain resilience strategies are effective, but a common tradeoff worth considering is that flexibility requires more effort to implement, while redundancy comes at a higher cost. Industry constraints may incentivize moving towards a certain approach, for instance, pharmaceutical companies have to keep regulatory minimum inventories for some products; regardless, hybrid approaches can also be implemented since these can be effective as studied by Simchi-Levi et al. [30].

A challenge when implementing supply chain resilience initiatives is selling the risk-reward tradeoff to stakeholders because it is not straightforward. In finance, any investor would expect to receive a higher payoff when investing in riskier assets. Inversely, in supply chain resilience, investing in reducing the impact on revenue of future risks would mean increasing costs and reducing profits in the present. Figure 3-1 represents how typically achieving resilience requires increasing costs, although opportunities exist to devise smarter alternatives that reduce risks without directly impacting profits.

Chopra [31] ponders on such a dilemma and shows how undermining disruption impacts results economically worse than over-allocating resources on resilience, but acknowledges the challenge of measuring resilience within an organization to bring stakeholders on board.

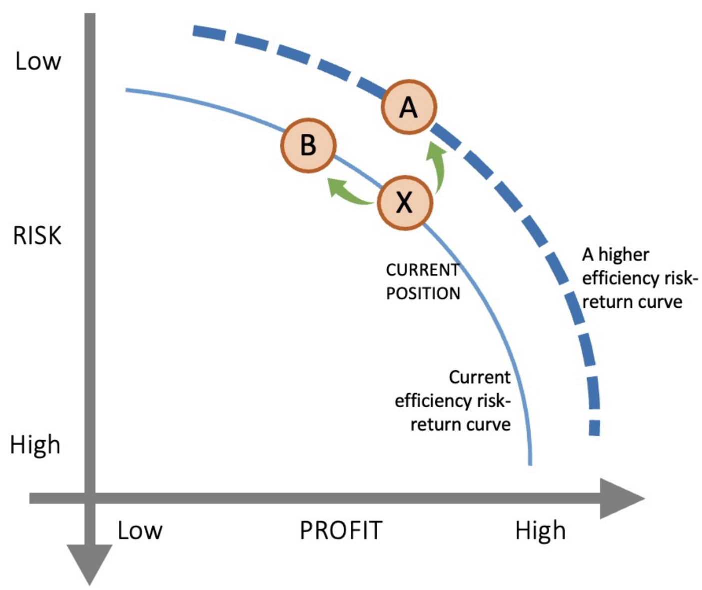


Figure 3-1: Choosing Supply Chain Risk/Reward Trade-offs [32]

Note: adapted figure from an exhibit of the original source

Simchi-Levi et al. [33] demonstrate how to overcome such a challenge by developing a novel model that relies mainly on two resilience metrics: (1) time-to-recover (TTR), *"the time it would take for a particular node (such as a supplier facility, a distribution center, or a transportation hub) to be restored to full functionality after a disruption"*

[34]; and (2) time-to-survive (TTS), *"the maximum duration that the supply chain can match supply with demand after a node disruption"* [35].

The model has been widely adopted across several industries as Simchi-Levi mentions [36]. Much of the model's success can be attributed to its ability to help organizations identify risks throughout their supply chains. Risks are determined by calculating and comparing the TTS and TTR at a node level. If a TTR takes longer than the corresponding TTS, the node requires attention.

To close the gap between the TTR and TTS, organizations could opt to increase inventories; however, Chopra [32] suggests that certain mitigation strategies may be more effective depending on the underlying risk at the node.

Consequently, the digital tool presented in this thesis intends to encompass features, described in Table 3.1, that allow organizations to identify risks in their supply chains and adopt a proactive supply chain resilience stance.

These features were selected considering that companies may want to implement redundant or flexible strategies, measure their end-to-end supply chain resilience, leverage the results to gain buy-in from leaders within their organizations, evaluate different response mechanisms, and recurrently evaluate their supply chain network risks.

Tool Feature	**Underlying Objective**
Configurable	Test any desired combination of disruptions and response mechanisms
Modular	Represent as many nodes and tiers in a supply chain as desired
Practical	Expedite the preventive and responsive decision-making processes
Universal	Applicable to any desired product and set of products
Visual	Support communication with stakeholders

Table 3.1: Ideal digital tool features to achieve resilience

3.2 Digital Twins

The idea of a DT centers around having a virtual environment that enables users to experiment with a representation of a system in the physical world in a setting without consequences. Such a virtual environment would be a safe space for learning and any insights gained could be applied to the physical entity, accelerating continuous improvement and reducing costs.

The idea of a DT, if materialized, unleashes countless applications for any industry. For instance, in the pharmaceutical industry companies could test therapies on digital organs instead of real patients to learn how those organs respond to medicines and modify the treatments to increase their effectiveness, reduce side effects, and shorten clinical trials to reach patients sooner.

Although the term digital twin (DT) has existed for more than two decades [37]; nevertheless, a proper definition accepted by the research community is yet to be agreed upon [38]. Unfortunately, the absence of a standardized definition of what a DT is, can lead to unintended misusage of the term among those unfamiliar with the core idea. The lack of common understanding, in hand with the DTs advancement, has made 'digital twin' a fashionable term - a buzzword.

Tozanli and Saénz [39] describe DTs as *"virtual replicas of physical entities and their interactions"*. This definition is succinct and adequately captures the essence that Grieves conceptualized [37].

Moreover, Tozanli and Saénz [40] provide three other facts - generally mystified - to further clarify the concept and to incentivize a wider adoption among corporations:

1. DTs consist of a "combination of enabling technologies and analytics capabilities" [40].

   Many current technologies are applicable for developing DTs, for instance: cameras, sensors, RFIDs, and LiDARs allow data collection from the physical entity; virtual reality, augmented reality, and 3D modeling software allow visualizing the physical entity in a digital environment; AI, machine learning, simulation, optimization, and business intelligence provide the analytics capabilities; and

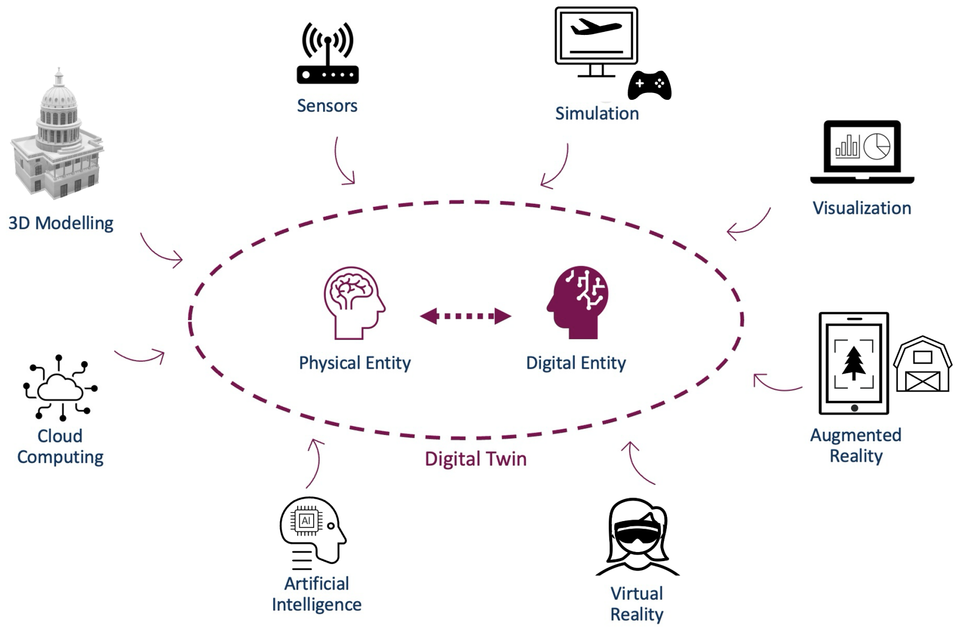


Figure 3-2: Conceptual model of a digital twin

cloud, edge or fog computing enable transferring between both entities. The technologies listed are not exhaustive but help illustrate why a combination is needed to fulfill the DT spirit.

2. The DT technology "has become more accessible and affordable" and is "compelling and deliver[s] value" in supply chain management [40].

   DTs are already in use across various industries. NASA utilizes DTs for autonomous in-space assembly [41], San Francisco to operate its airport [42], Rolls-Royce to predict engine behavior [43], Renault to manage its manufacturing lines [44].

   For supply chain management, the opportunities are also vast and present. Tozanli and Saénz highlight examples for supply chain planning, warehouse management, and transportation management [39].

3. DTs "can be created before its equivalent physical asset is built or acquired" [40].

> Every design process goes through a prototype phase before launching a product to the market, as shown in Figure 3-3, and developing a DT is no different. Kritzinger et. al [45] describe three levels of integration of DTs: (1) digital models - the stage where the changes of one entity do not affect its twin and data exchanges occur manually; (2) digital shadows - the stage where changes in the physical entity translate automatically to the DT, but not the other way around; and (3) digital twins - the stage where both entities are fully connected and improve each other automatically.

Understanding what a DT is and what it is not becomes crucial when implementing the technology. It helps define a better use case, design a more reasonable action plan, and build a stronger business case to convince stakeholders.

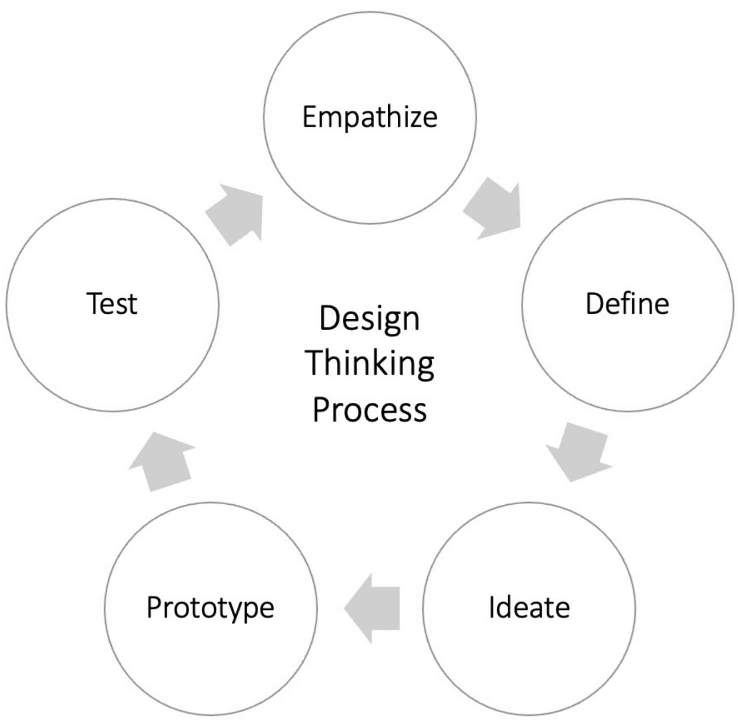


Figure 3-3: Design Thinking Process [46]

Chapter 4

Methodology

4.1 Approach

The thesis project addressed a dual challenge. On the one hand, it involved developing an in-house digital tool for AZ to stress-test the supply chain resilience of their brands and define BCPs. On the other hand, the tool needed to serve as a building block for a supply chain digital twin.

The essence of the problem was to create a product with existing company software that would deliver business value to stakeholders. As Brown explained, design thinking can be a powerful technique for solving those desirability, feasibility, and viability challenges [47]. Hence, the approach taken to develop the tool was design thinking, a problem-solving methodology consisting of five stages:

1. *Empathize*: this stage enabled uncovering opportunities, limitations, and expectations.

2. *Define*: this stage enabled narrowing the scope and breaking down the problem into manageable activities.

3. *Ideate*: this stage enabled exploring ways to add value.

4. *Prototype*: this stage enabled the tool creation according to users' needs.

5. *Test*: this stage enabled the tool validation for business applications.

Since design thinking is a human-centered approach, a Steering Committee was formed to reinforce the methodology. The Steering Committee comprised members from GSC&S and ESM, including the pmdiTech brand director. Having such a group of stakeholders was fundamental for reaching a consensus on key decisions, obtaining constant feedback and guidance, and increasing the project's likelihood of success.

Subsequent sections will delve into the specific activities, challenges, and insights of each stage.

4.2 Empathize Stage: Understanding Stakeholders

Empathizing required perspective-taking. To achieve this, the first thing to do was to segment the stakeholders involved into groups according to their connection with the tool and arrange individual meetings with all of them.

The *first group* consisted of the direct users of the tool, starting with pmdiTech's brand director and other stakeholders who would benefit the most from the project. Members of this group belonged to GSC&S, the functional area that owned pmdiTech's P&L and BCP. Their strong interest in the project's outcome provided crucial insights.

The most valuable lesson was understanding the BCP process[1]. The objective of the process was to evaluate if a node could survive an outage of a certain pre-defined length. If the node couldn't survive, the GSC&S team along with the corresponding ESM manager, would evaluate potential corrective actions such as increasing inventory, investing in backup tools, or implementing dual sourcing. To calculate if a node could survive, they employed a spreadsheet to perform arithmetic operations on inventory policies, lead times, and outage durations.

The process was manual, tedious, and did not provide much value; it was then clear why the project existed. Moreover, the group's initial expectations of the tool were to measure the impact of demand surges or shortages and test the effectiveness of the potential corrective measures. A challenge to building a more comprehensive

[1]The process description intends to provide a high-level overview, but it does not precisely reflect the way AZ works

model was gathering data due to the absence of a value stream mapping (VSM) for the external supply chain. A positive aspect was that the team was also responsible for generating demand forecasts for both the short and long term, so they could facilitate the information if needed.

The *second group* consisted of the leaders of AZ's Operations department, the stakeholders with sufficient authority to endorse the tool and its progression beyond the POC phase. Reaching out was challenging given their busy schedules, but interacting with them was essential to create awareness of the project's existence since they would advocate for adopting the tool at an organization-wide level. This group made clear their interest was in the tool's result rather than the development process.

The *third group* consisted of the ESM managers, the stakeholders who were indirectly affected by the tool given their involvement in the BCP process. Members of this group handled the relationship with suppliers and owned information about pmdiTech's external supply chain, such as the number of lines at each site, the capacity of each machine, the manufacturing schedules, the investment timeline for acquiring new equipment, the overall equipment effectiveness (OEE), and more; however, instead of having a central repository for the data, it was spread out in files; there was no single source of truth.

This group's deep understanding of the manufacturing processes, their close relationship with the suppliers, and the information they held made them extremely valuable for developing a successful tool. Interacting with them revealed their interest in being involved in the project and an opportunity to expand the tool's use case.

The *fourth and final group* consisted of members from the IT department, the stakeholders with very little interest in the project but who had vast experience in digital developments. Members of this group were already working on DTs for other use cases, so the goal of the conversations was to understand if there was a conventional method within the company. Three extremely valuable insights emerged: (1) using external tools[2] required authorization and a lengthy validation process; (2) the company was only working on product DTs, so if the tool met the POC standards,

[2]Not found in the AZ Software Store

it would become the first process DT in the pipeline; (3) a digital tool needed to satisfy a business case and had financial support from their department leaders to gain POC status and, then, the tool would be scaled by IT within a 6-12 month period, with the caveat that the IT department would only use it to understand how it worked but they would build a new solution from scratch.

The Empathize Stage demonstrated that interviewing stakeholders provides valuable insights and generates interest regardless of their connection with the project. It also provided the minimum expectations and standards for the project to be deemed successful. It further highlighted the opportunities to improve the current BCP process and the limitations regarding data availability. Finally, it allowed for understanding that the path to a digital twin POC was to think big but start small.

4.3 Define Stage: Clarifying Objectives

Defining sought to narrow down the problem in scope. After several weeks of empathizing with stakeholders, some characteristics of the tool were already known, but the problem was still very broad.

On the one hand, the tool was intended to measure the resilience of the supply chain under stress scenarios, but neither the types of supply chain disruptions the company wanted to assess nor the metrics to quantify the performance impact - crucial for senior stakeholder engagement, as discussed in Chapter 3.1 - had been established. In addition, the goal of including all the nodes in the pmdiTech supply chain in the model was still very ambitious due to the limited access to information, so it was necessary to prioritize the nodes to be included in the initial model and gradually add them to the model.

On the other hand, the tool was intended to be the basis of a digital twin, but the enabling technologies that would be part of it had not been selected, as outlined in Chapter 3.2. At the same time, since there was the limitation of working only with pre-approved software, a survey was conducted to classify the most important features expected from the tool, such as intuitiveness of use, cost per user, and speed to obtain results, among others.

The aforementioned points were discussed at the Steering Committee and the initial agreements were as follows:

- *Disruption types*: tool malfunction, machine breakdown, site closure, demand upsurge, region-wide disruptions (i.e. limited energy usage across EU)
- *Performance metrics*: impact on revenue
- *Resilience metrics*: time-to-survive and time-away-from-design - new concept
- *Supply chain scope*: from formulation to packing, including tier-one suppliers for devices, assembly and packing (refer to Figure 2-2)
- *Nodes to map*: AZDP and two device suppliers
- *Enabling technologies*: simulation and business intelligence
- *Software features ranking*: intuitive user interface (most important), free, customizable views, response time (least important)

The Design Stage proved useful for making complex decisions in an unbiased manner and considering the users' needs. It further set the project boundaries, simplifying the problem. It paved a clear path for gathering data and brainstorming possible solutions.

4.3.1 Data Collection

After reaching a consensus on most aspects of the tool and a newer scope, it became possible to start gathering data - a crucial activity for obtaining reliable results. Although the data collection process started immediately after the Define Stage, the activity progressed slowly, overlapping with the Ideate and Prototype stages.

Before collecting information, the first step was to *create a list of the key data* for developing the tool. These items were selected mainly based on the Simchi-Levi et. al TTR and TTS models [33], although items identified as potentially valuable in the

Empathize Stage and others suggested by Simchi-Levi et. al [34] were also included. Table 4.1 shows the summary of the items collected and ended up being used by the tool.

The second step was to obtain *information from internal sources*, with the help of ESM managers, and minimize the time suppliers would spend collecting everything requested. Finally, formal approaches were made to suppliers to obtain the remaining information and validate some of the items already collected.

Requesting information from third parties presented challenges, as data sharing could be perceived as intrusive and potentially harmful to business relations.

To *persuade suppliers*, meetings were arranged at their manufacturing sites. During the visits, Gemba walks were key to understanding the logical order of the manufacturing process of the device and to validate some data points that had already been identified.

Additionally, these visits served to discuss business matters and the project's potential operational advantages, highlighting how digital twins had been effectively utilized by other organizations. A particular emphasis was placed on the fact that the project would ultimately benefit patients, streamlining the collaboration given the common mission across supply chain entities in the pharmaceutical industry.

Following supplier alignment, *questionnaires* were distributed to capture any outstanding data. Surveying through questionnaires required precise language and illustrative examples, often necessitating additional clarification to interpret the results accurately.

Moreover, another powerful source of data was to *review contracts* of the tier-one suppliers. This was not considered until later in the project, but it allowed collecting data from any tier-one supplier and not just the ones in scope. Contracts accelerated the data collection because several times minimum or maximum inventory policies, lead times, minimum order quantities, or even service level agreements were specified there; however, if contracts prove to be out of reach, the methods described above demonstrated being useful. Furthermore, reviewing contracts in advance could have facilitated identifying risky suppliers during the Gemba walks.

Item	Description
BOM	For the product and components
Demand	Consumer-only data
Inventory Policy	Bookmarked for the company and the brand in scope, at each supply chain stage (raw materials, WIP, finished goods)
Installed capcity	Theoretical machine capacity dedicated for the company, assuming 24/7 operation at 100% OEE
Machine availability	Operating schedule (e.g., 24/5, 24/7)
OEE	Current and projected, considering machine age and ramp-ups
Minimum order quantity	As required by tier-one suppliers
Lead times	For manufacturing and delivery
Scheduled downtime	Holidays, annual shutdowns
TTR	Estimated time range for recovery from disruptions requiring tool, equipment, or site replacement
Additional resources	Extra capacity options (e.g., extra shifts, shared machines)
Location	Of tier-one and tier-two suppliers
Capacity utilization	Average per machine

Table 4.1: Key data collected and used by the final version of the tool

4.3.2 Metrics

Performance Metrics

To measure the impact on revenue the tool kept track of the on-time and late deliveries of each product across the simulation period. The on-time deliveries were then multiplied by the price of each product and the total revenue across the simulation was displayed on the visualization dashboard. Users could then filter across scenarios and compare the results to calculate the impact.

To complement the metric, the costs of goods sold were also calculated and displayed in the visualization dashboard of the tool. The tool multiplied the cost of each component by the number of components purchased by AZ during the simulation.

Resilience Metrics

Given the limited access to data, building an accurate optimization model within the tool, such as the TTS model suggested by Simchi-Levi et al. [33], proved challenging.

Moreover, the objective of the tool was not specifically to find the best solution given a set of constraints.

Instead, developing a tool with simulation capabilities added a time dimension to the analysis that facilitated understanding the system's behavior under different stress conditions and facilitating comparisons.

Adding visualization capabilities to the tool on top of simulation yielded a powerful tool for analysis, especially in the context of dealing with complex systems and metrics.

Thus, visualizing the evolution of the inventory coverage of each component instead of calculating the actual TTS was deemed appropriate among the Steering Committee.

The tool calculated the inventory coverage as the inventory on hand at the beginning of the period divided over the weekly estimated demand for the component in scope. The weekly estimated demand was the forecasted demand for the product over the following 12 months divided over 52 weeks. Given that the demand was only forecasted for the final products of the supply chain the calculation started there and rippled to the rest of the components required for such product according to the BOM.

The company also wanted to understand how long inventory stayed below the baseline scenario - the time-away-from-design. Since the concept was not a metric in itself, the idea was also to provide visualizations that allowed users to visually identify the concept.

Consequently, the focus shifted to developing visualizations that allowed users to easily interpret data and informed decision-making.

4.4 Ideate Stage: Conceptualizing Solutions

Ideating called for thinking big within a narrow scope and in compliance with some constraints. After the Define stage, the project objective became to develop an in-house tool that allowed AZ to stress test the supply chain resilience of their brands for defining BCPs, leveraging simulation and visualization capabilities. This entailed two primary tasks: (1) designing the tool's logical framework and selecting appropriate software to fit such a framework; (2) conceptualizing a model for simulating supply chain operations and disruptions.

4.4.1 Tool Logical Framework

The first activity of the Ideate Stage required conceiving a logical framework of how the tool could solve the general objective. A concept of this logic, shown in Figure 4-1, would be as follows: users would collect data and upload it to a data repository, the tool would take the data from the repository, run the simulations, and display the results, enabling the user to define a BCP, run more scenarios for further analysis, or collect more data to expand the model before starting over. A challenge with this framework was finding software that allowed users to modify scenarios in the front end of the tool without accessing the back end of the tool.

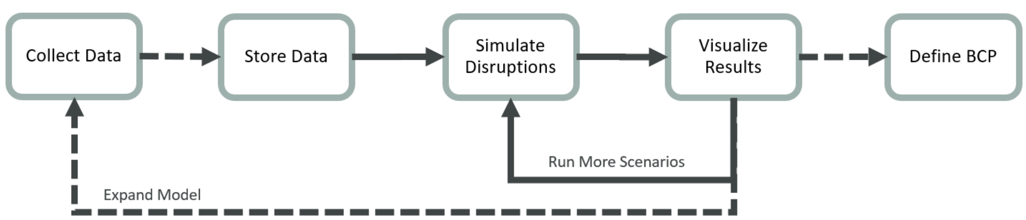


Figure 4-1: Initial conceptual framework of the tool logic

Note: dotted lines represent manual tasks

Python was chosen as the main engine of the tool due to its widespread use within the organization, its versatility, and its relative simplicity compared to other languages. Python added the simulation capabilities to the tool but lacked visualization. Integrating Python with some libraries[3] like Dash or Streamlit would have made the solution technically feasible, but a mock version of a dashboard created with one of those visualization libraries raised data privacy concerns and was unfamiliar to the Steering Committee.

As the tool's responsiveness was not critical, since the tool would already enhance and simplify the BCP process, an alternative was to keep Python as the tool engine, using Visual Studio Code as the development environment, and pair it with Power

[3]Collections of predefined functions.

BI or Tableau for visualization purposes. Both business intelligence solutions were equally intuitive, but Tableau did not meet the cost requirements and stakeholders were generally more familiar with Power BI. Hence, the software combination for the tool resulted in Python for the simulation and Power BI for the visualization, but two other problems arose.

Python could not directly export the results to Power BI, while Power BI could not directly alter the disruption parameters and initiate a Python script. To solve the first half of the problem, Python would export the results to a spreadsheet in a predefined layout, so the user could only refresh the Power BI dashboard and visualize the latest results. The second half of the problem was more complicated and deemed not worth solving given that the tool was in a POC phase, and the value added would come from its business applications.

To keep the solution streamlined and effective without adding more software components, spreadsheets were also used as a repository for data. However, this implied that after getting the simulation results, the user had to manually adjust the spreadsheet before simulating more scenarios.

Consequently, the tool consisted of a combination of three software: Python, Power BI, and Excel. The user-facing part of the tool was called the Front End, while the computational aspects were designated as the Back End. The final logical framework of how the tool could be applied for BCP purposes is shown in Figure 4-2.

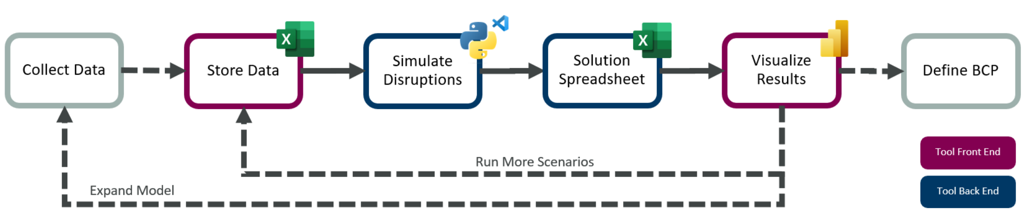


Figure 4-2: Final conceptual framework of the tool logic

Note: dotted lines represent manual tasks

4.4.2 Supply Chain Model

The second activity of the Ideate Stage required modeling a logic to simulate the supply chain. Since a model is just a representation of how something works, the supply chain model was built upon the subsequent three observations:

1. *Source, Make, and Deliver can be found anywhere in the supply chain.* Any node in any supply chain can perform three main processes: order materials from a supplier (Source), transform the materials into products (Make), and deliver the products to its customers (Deliver). The idea holds within a node; processes are connected in the same way.

2. *Demand triggers the flow of products in the supply chain.* For instance, fulfilling a demand order reduces finished goods inventory, creating a manufacturing order to replenish the lost inventory, and further creating a source order (demand for the tier-one supplier) to replenish the raw material required to manufacture the goods, and so on. The idea holds if the node does not have inventory policies.

3. *Each node behaves differently.* The first two observations dominated the behavior of the overall supply chain, but nodes have specific characteristics that regulate their behavior. No two nodes are identical, and the same idea applies to processes.

Those three observations helped depict a conceptual model of the supply chain, as seen in Figure 4-3.

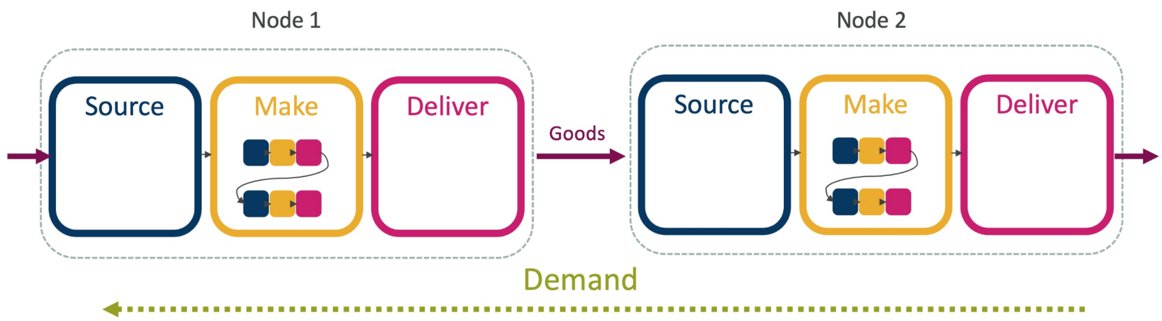


Figure 4-3: Conceptual model of the supply chain

4.4.3 Supply Chain Model Boundaries and Assumptions

Including all the specific behaviors of each node was not possible nor was it the intention of the model - it would have defeated the purpose of developing a tool applicable to more than just the brand in scope. Hence, making reasonable assumptions and setting boundaries was necessary to avoid going down a neverending spiral of specifics. Each assumption and boundary is explained in detail below:

1. *Boundary: shipment capacity and allocation were out of scope.* The problem focused on manufacturing capacity, not on transportation capacity. Although disruptions could impact vehicle availability or mobilization, it was easier to replace such a service than to find more manufacturers with regulatory approvals to produce medical devices. Thus, Deliver was the least detailed process.

2. *Assumption: unconstrained supply from nodes not included in the simulation.* Not every node was going to be included in the model at once because information was not available for the end-to-end supply chain across all tiers. Nodes lying outside of the model boundaries were not measured and should not have influenced the model.

3. *Assumption: orders were placed at a weekly level.* Using a weekly granularity to represent the time frame at which processes occurred was consistent with the time horizon of the data collected, for example, inventory policies were measured in weeks of supply and inventory coverage also used weeks. A more specific time frame would have increased the computational time of the model without adding much value when analyzing the results, while modeling processes at a more aggregate time frame would not have represented the processes adequately.

4. *Assumption: orders were automatically received and accepted.* This implied that processes within a node were coordinated or even connected through systems. Moreover, it was considered that node-to-node communication during a disruption would increase; therefore, it was not worth detailing how each node accepted or rejected orders.

5. *Assumption: orders were prioritized by due date.* The underlying intention of not stocking out was to maximize service level; therefore, the focus was on reducing late deliveries. Backlogs had to be resolved before working on new orders and orders were never cancelled.

6. *Assumption: production occurred as soon as possible.* Nodes did not intend to incur any unnecessary delay. This also assumed that the holding cost for raw material and finished goods were not too different to change the behavior of a node.

7. *Assumption: inventory never went obsolete.* Obsolence had not been a problem before. Most products in scope were not perishable and the expiration date of those that were was much later than what inventories kept would last. For example, holding six months of inventory of a product with an expiration date of two years does not impact.

8. *Assumption: warehouses had infinite capacity.* Warehouse capacity could have been incorporated into the model; however, losing such a constraint allowed illustrating visually the volume impact of changing a policy, especially towards the future as demand grew.

9. *Assumption: partial deliveries were allowed.* Prioritizing responsiveness was considered more important in the model than the transportation cost.

10. *Assumption: deliveries never arrived ahead of time.* Adding this assumption was required to balance the previous one. Supply chains would try to benefit from economies of scale, but disruptions would potentially modify such behavior.

4.4.4 Stress Test Considerations

Stress testing scenarios was a main component of the project's objective. Recalling the discussion in Chapter 4.2, the original BCP process only evaluated node outages and did not test any corrective mechanisms. To enhance the BCP process, more alternatives had to be provided both for creating disruptions and taking action.

To model supply chain disruptions, the examples discussed during the Define Stage were grouped according to the consequence it would generate on the node directly affected. Three categories resulted: capacity disruptions, inventory disruptions, and demand disruptions. Moreover, an opportunity to include the option to affect lead times was considered, but such an idea was left out since the delivery process had been deemed out of scope for the project.

Meanwhile, the response mechanisms included in the model followed were related to increasing capacity or adding inventory, both redundant approaches. The four policies considered for the tool were the following: changing inventory policies, changing a machine's availability (i.e. shifts), changing a machine's OEE, or adding a dual source.

The Prototype Stage would then try to configure the tool to allow stress testing of any number of disruptions and any combination of disruptions and response mechanisms shown in Table 4.2.

Item	**Type**
Disruptions	Capacity
	Inventory
	Demand
Response Mechanisms	Changing inventory
	Changing a machine's availability
	Changing a machine's OEE
	Adding a dual source

Table 4.2: Stress Test Considerations

This table and the preceding assumptions guided the development of the tool's capabilities to simulate and analyze different stress scenarios effectively.

4.5 Prototype Stage: Tool Building

Prototyping sought to develop the Front End and Back End components of the tool, while trying to incorporate the tool features identified in Chapter 3.1: configurable, modular, practical, universal, and visual.

4.5.1 Tool Back End

The first piece of the puzzle required translating the supply chain model into a simulation code. To simulate the conceptual model of the supply chain, a discrete event simulation approach was deemed appropriate. A goal throughout the code development process was to design a script that enabled modularity and reusability.

Code Main Logic

The main logic of the code was built upon the observed supply chain behaviors; therefore, at each period of the simulation: (1) each node would Source, Make, and Deliver; (2) a demand order would be generated and the most downstream node would receive it; (3) disruptions would modify how certain nodes behaved during the simulation period. Those actions were embedded in a larger loop, allowing users to test multiple disruption scenarios simultaneously. At the end of each disruption scenario, the results were recorded, and after completing all disruptions the results were exported to the Solutions Spreadsheet (refer to 4-2).

The main logic described was represented in the mock code below. For the actual code described in this section, please refer to Appendix B.

```
for scenario in Disruption Scenarios:
    for period in Simulation Length:
        Disrupt
        Generate demand order
        for node in Nodes:
            Source
        for node in Nodes:
            Make
        for node in Nodes:
            Deliver
    Record results
Export results to Excel
```

While writing down the logic, it was observed that each Source, Make, and Deliver process had to occur ‘simultaneously’ for all nodes but these processes would take place sequentially. That meant that orders would be placed at the start of the period, nodes would work on outstanding orders throughout the period, and goods would be delivered at the end of the period. Otherwise, there would have been a lag of one period in the production, for instance, a node went through the Make process without necessarily having the most updated orders from its customer or without having received raw materials from its suppliers.

To execute the logic it was necessary to represent objects and write functions. The use of external libraries was kept at a minimum to facilitate reading and maintenance.

Code Objects

Objects in the code were considered the elements of the logic that carried particular data - attributes. Classes were used to represent instances of objects, enabling tracking changes in the attributes of each object instance throughout the simulation, and providing the necessary flexibility and modularity to make changes as the code was written.

Each class stored the data of each attribute using different data structures. Moreover, each class also had small functions - methods - to facilitate its use, although those are not relevant to the discussion.

A class was created for each of the following objects: nodes, products, processes, orders, and disruptions. The attributes of each class are summarized in Table 4.3.

Nodes represented all the sites in the supply chain. Nodes were a robust class because they contained classes of products, processes, and orders. Through a node, it was possible to determine what processes had to be followed to create a process, when to place orders, and how much to request from suppliers. Leaf nodes - nodes that were at the boundary of the supply chain (i.e. consumer nodes) and were included in the model - did not necessarily use all the attributes but did use the same class. A special list kept track of the leaf nodes at the upstream boundary to exclude them from sourcing, while consumer nodes only played a role in the demand generation part

Class	**Attributes**
Node	Name, raw materials, finished goods, bill of materials, processes, process sequence, holidays, orders
Product	Name, customers, suppliers, inventory, design stock, inventory gap, annual demand, inventory coverage, late deliveries, replenishment time
Process	Name, inventory, design stock, inventory gap, annual demand, inventory coverage, capacity available, orders, machines
Machines*	Production, utilization, availability, OEE, capacity, availability, capacity available, capacity threshold, percent available, start date, end date
Order	ID, status, internal (created within node), date received, supplier, customer, product, demand, date due, process, amount produced, date production starts, process amounts required, date production end, amount shipped, date shipped, amount delivered, date delivered, date fulfilled
Disruption	Start date, end date, node name, product name, process name, machine name, capacity threshold, inventory loss, design stock, demand variation, machine availability, machine OEE, type

Table 4.3: Summary of objects in the code

*Machines are not a general class, but data was tracked for individual machines

of the code.

Products represented either finished goods, raw materials, or work in progress (WIP) products in between production steps. Accounting for WIP products was necessary because sometimes the flow of products did not occur within the same site. Moreover, products kept track of their own inventory over time, as well as the inventory coverage at each period and the accumulated late deliveries.

Late deliveries were tracked by the model so users could determine if a portion of the unmet demand would be lost for subsequent periods, penalizing long-term revenue. A harsh penalty, but potentially true depending on the brand and the country, given that doctors would just prescribe another brand or governments would find other suppliers.

Processes represented all the activities performed at a certain node. In the model, processes kept track of how many WIP products had finished the activity but had not been taken by the subsequent process, avoiding the need to create a product for each WIP part.

Processes also contained machines. Depending on the amount of data collected, users would be able to track each manufacturing line instead of the process as a whole, allowing for testing disruptions on specific equipment if desired. Machines were not classes per se, but did keep track of several statistics about utilization.

Orders held information about the product, customer, and supplier involved in the transaction. Although these items were just meant to trigger the supply chain flow, it was also possible to use them for tracking partial deliveries and other statistics for analysis.

Disruptions served two purposes. First, to keep track of when a disruption would occur, how long would it last, what type of disruption it was, and what nodes would be affected. Second, to keep track of the corrective measures users wanted to test. However, it was not possible to have an instance of this class that was both a disruption and a corrective measure at once, multiple instances had to be created. Using classes to represent disruptions and response mechanisms also facilitated applying multiple disruptions within the same period.

Code Main Functions

The interconnectivity between classes facilitated building functions that required very few arguments. The main functions of the code represented the activities of the general code logic: disrupt, source, make, and deliver. Generating demand was not a function per se, because it was a much more simplified version of Source and it was directly coded in the main logic block of the code.

The *Disrupt* function occurred once per period. The function took the list of all the disruptions and iteratively checked, for each disruption, if the start or end date matched the simulation period. If it did, then the function would check the type of disruption, check which nodes had to be affected, and modify an attribute of the node depending on the type of disruption.

A capacity-type disruption would modify the capacity threshold of a machine or process, limiting the capacity available to a certain percentage of the capacity installed; an inventory-type disruption would reduce the inventory of a product or process by a certain percentage; a demand-type disruption would intensify or weaken a consumer demand by a certain percentage; a policy-type disruption, a response mechanism, modified the design stock of a product or process; and a machine-type disruption, a response mechanism, changed the availability or OEE of a machine.

A representation of the function logic can be seen below:

```
for disruption in disruptions:
    if simulation period = disruption.start_date:
        if disruption.type = 'capacity':
            Find Node
            Reduce Capacity Threshold
        elif disruption.type = 'inventory':
            Find Product or Process
            Reduce Inventory
        ...
    elif simulation period = disruption.end_date:
        if disruption.type = 'capacity':
            Find Node
            Capacity Threshold = 1
        elif disruption.type = 'demand':
        ...
```

The *Source* function was applied once per period to each node except consumer nodes or leaf nodes. The objective of the function was to determine if orders needed to be placed, either for raw materials, finished goods, or to WIP inventory.

The function calculated the inventory gap for each product as the difference between the target and actual inventory levels, adjusted by the pending inflows and outflows of the product. The target inventory level was the design stock, in weeks of supply, multiplied by the average weekly demand of the next 52 weeks. The on-hand inventory level was the inventory at the end of the previous simulation period. The pending outflows were the product quantity allocated to outstanding orders from customers, while the pending inflows were the product orders that suppliers had not delivered yet. A positive result would trigger an order, otherwise nothing would happen because the node had excess inventory.

Furthermore, in the case of raw materials, the function divided the orders among suppliers based on their capacity, inventory, and minimum order quantity for the raw material. The function tried to source from as many suppliers as possible, as long as the split amounts met the minimum order quantity required. While doing so, the function tried to balance the order allocation among the suppliers, according to their proportion of the total capacity and inventory available.

A representation of the function logic can be seen below:

```
for raw material in node.raw_materials:
    Calculate inv required
    if inv required > 0:
        Define order split
        Assign orders to suppliers
for finished good in node.finished_goods:
    Calculate inv required
    if inv required > 0:
        Create internal production order
for process in node.processes:
    Calculate inv required
    if inv required > 0:
        Create internal production order (partial)
```

The *Make* function was applied once per period to each node. The objective of the function was to produce as many products as necessary according to the outstanding orders but limited by the inventory available at the beginning of the period and the capacity available.

To validate if an order would be produced, the function calculated how many raw material units were necessary to fulfill the order, considering the bill of materials, and compared it to the inventory available of such raw material. If there was enough inventory, the function then verified if the machines within the process had enough capacity available for production during the simulation period. The function then maximized the production and adjusted the order status, the raw material and finished goods inventory levels, and the machine production.

A representation of the function logic can be seen below:

```
for order in outstanding_orders:
    Assign order to processes
for process in manufacturer.processes:
    for order in process.orders:
        if inventory available > 0:
            if capacity available > 0:
                Reduce raw material inventory
                Increase finished goods inventory
                Update order status
                Update machine production of period
```

The *Deliver* function was applied once per period to each node. The objective of the function was to determine when to ship finished goods from a supplier to a customer according to the inventory available and the delivery lead times, ensuring timely deliveries.

The function first identified the orders that needed to be shipped based on the due date and the delivery lead time. Then, it shipped the orders according to the available inventory and the remaining demand. Afterward, it updated the shipped order status and the supplier's inventory. It then checked the orders that were in transit and delivered them to the customers, updating the customer's inventory and the order status.

A representation of the function logic can be seen below:

```
for order in supplier.orders['not␣shipped/delivered']:
    if sim period >= order.date_due - delivery_lead_time:
        Reduce inventory at supplier
        Update order status
for order in supplier.orders['in␣transit']:
    if sim period - delivery_lead_time = order.
       date_shipped:
        Increase inventory at customer
        Update order status
```

Code Auxiliary Functions and Solution Spreadsheet

Auxiliary functions were required to initialize objects, reset the objects in between each disruption scenario, record results, and update the expected demand on a rolling basis.

Furthermore, the code had some blocks to read the data from the spreadsheet where the data was stored, to create the instances of each object, and to print the outputs into the Solutions Spreadsheet.

Leveraging Pandas[4] allowed importing and exporting data from spreadsheets. To export the simulation results, the data contained in the classes was captured in Pandas DataFrames[5]. Those DataFrames were printed in the Solutions File across three tabs:

1. *Production Tab.* Included a column for: Date, Scenario Name, Node, Process, Machine, Machine Utilization (units), Machine Utilization (%), Capacity Available (units), Capacity Available (%), Machine Start Date, and Machine End Date

[4] An open source Python library for manipulating data efficiently.
[5] A tabular data structure with labeled rows and columns, similar to the one in spreadsheets.

2. *Inventory Tab.* Included a column for: Date, Scenario Name, Node, Type (raw material, WIP, or finished goods), Component (product), Design Stock (weeks of supply), Inventory (units), and Inventory Coverage (weeks of supply)

3. *Orders Tab.* Included a column for: Scenario Name, Order ID, Node (customer), Supplier, Component (product), Internal (boolean), Demand (units), Production (units), Date Placed, Date Production Started, Date Production Ended, Date Fulfilled, Due Date, Date Shipped, Amount Shipped (units), Date Delivered, Amount Delivered (units), Late (binary)

4.5.2 Tool Front End

So far the Back End of the tool encompassed the modular and universal features desired for resilience. The second piece of the puzzle required developing a Front End that reinforced those features, while incorporating the other three ideal attributes: configurable, practical, and visual.

Input Template - Data Storage

Building the Input Template after having a code structure was not complicated, but it was tedious. It was necessary to design the tool in an interactive and user error-proof way, which required adding macros, conditional formatting, dynamic dropdown lists, data validation rules, a README tab, and protecting cells.

The first step was to create a tab that would allow users to add or remove nodes, processes, or products to the model quickly and easily. This tab dubbed *Reconfigure Supply Chain*, shown in Appendix C-1, included three buttons: Node, Process, and Component (product). Naturally, each button had the option to add or delete; however, the Process and Component buttons had to be added to a node. Dropdown menus were added to select from the existing list of items and errors would pop up if an invalid action was performed, such as removing items that did not exist or duplicating names.

The macros on the Reconfigure Supply Chain tab would fill another tab with the keys for each node, process, and product. These key tabs were hidden from the users and enabled adding the data validation rules and conditional formatting to the rest of the file. In every other tab, wherever a node, product, or process, was required, dropdown lists were available and updated automatically if the supply chain was reconfigured. Figure 4-4 displays an example of how these macros and the validation rules worked from a user perspective.

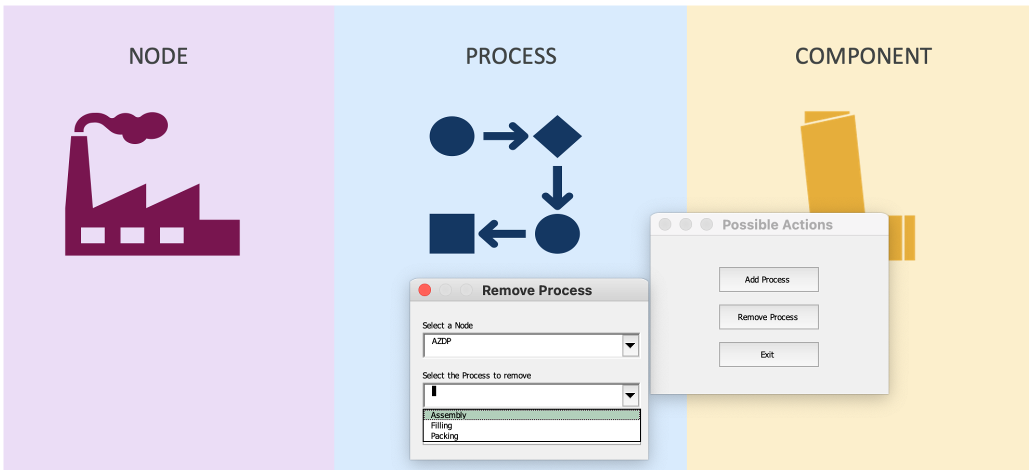


Figure 4-4: Example of how to reconfigure the supply chain

The second step was to create a set of nine tabs where users could modify the characteristics of the supply chain as necessary, such as the availability of a machine or the initial inventory of a raw material. Almost all these tabs followed a similar format: the first row specified the column header and each subsequent row was used to enter data. To facilitate adding rows while keeping the conditional formatting and validation rules, a macro button was included at the top of each tab. Cells that required values had data validation rules to ensure the data complied with the criteria, such as keeping percent values between 0 and 1, prohibiting negative values, allowing only integer values of inventory units, and more.

The *Processes* tab, shown in Appendix C-3, defined the processes contained within a node. This tab required: selecting a node and a process from the dropdown lists; determining the initial inventory units held at the end of the process; and specifying the design inventory in weeks of supply. Furthermore, this tab would flag whenever a user added an inventory value greater than zero to a leaf process, an ending process, by cross-referencing the BOM tab; instead, the inventory should have been inserted in the Finished Goods tab.

The *Machines* tab, shown in Appendix C-4, defined the manufacturing equipment for each process within a node. This tab required: selecting a node and a process from the dropdown lists; adding a name for the machine; determining the installed capacity in units per day, the availability in days per week, the OEE in percentage; and defining the start and end date of the machine, the time interval during which the machine was intended to exist in the supply chain. This tab would flag if machine names were duplicated.

The *Machine Updates* tab, shown in Appendix C-5, kept track of future changes in a machine's attributes to affect the simulation accordingly. This tab required: selecting a node, a process, and a machine from the dropdown lists; redefining the availability in days per week and the OEE in percentage; and the date that changes would become effective.

The *Raw Materials* tab and the *Finished Goods* tab, shown in Appendix C-6 and C-7 respectively, defined if a product was sourced (raw material) or manufactured (finished good) by a node. Both tabs required: selecting a node and a component (product) from the dropdown lists; determining the initial inventory units held at the node; and specifying the design inventory in weeks of supply. The Raw Materials tab would flag whenever a component was selected but the node did not appear in the Suppliers tab as a customer for such component. The Finished Goods tab would flag whenever a component was added to a node but the component did not appear as a finished good for the node in the BOM tab.

The *Suppliers* tab, shown in Appendix C-8, managed the relationship between nodes through the products exchanged. This tab required: selecting a node for the customer, a component (product), and a second node for the supplier; determining the minimum order quantity in units; and defining the delivery lead time in weeks. This tab would flag whenever a component was not in both, the Raw Materials tab for the Customer node, and in the Finished Goods tab for the supplier node.

The *BOM* tab, shown in Appendix C-9, managed the relationship between components and processes within a node. This tab required: selecting a node and a finished good from the dropdown lists; selecting either a required raw material or a required WIP component from their respective dropdown lists; determining the quantity required from the raw material or WIP component to produce the finished good; and selecting from a dropdown list the immediate process that required the raw material or WIP component. This tab would flag whenever a finished good and the raw material required were not in their respective tabs for the node selected, and it would also black out the required raw material or required WIP component cell if the other cell in the same row was occupied.

The *Demand* tab, shown in Appendix C-10, contained the monthly estimated values for each consumer (i.e. a country). This tab allowed the user to add as many rows (months) and columns (consumer) as desired. The Back End of the tool would determine the length of the simulation according to the number of rows; however, a column was included to allow users to determine if the simulation should end at a specific period without the need to delete rows, which also helped reduce the simulation runtime. The demand forecasts for the long term were disaggregated at a monthly level from an annual estimate according to the seasonality observed in previous periods to represent a more realistic behavior than simply using the average demand.

The *Holidays* tab, shown in Appendix C-11, tracked how many days of each week of any given year a node would shut down. Capturing this information would also enable modeling more accurate behaviors from each node.

The information in these nine tabs produced the Baseline Scenario of the model - a scenario without any disruptions. The third and last step was to create a tab where users could specify the disruption scenarios desired.

The *Disruption Parameters* tab, shown in Appendix C-2, was the most important tab of the file. The general idea of the tab was that each row would represent a disruption or a response mechanism. To group disruptions/response mechanisms within a scenario, users would only need to repeat the scenario name across the desired rows. To facilitate adding or removing disruptions/response mechanisms from the model without removing the row from the tab, a boolean column was embedded in the tab.

To guide the user through the process of adding disruptions/response mechanisms in the Disruption Parameters tab, each row would highlight in yellow the cells that required a value. First, the user would select the start date of the disruption/response mechanism, choose a node to disrupt from a dropdown list and determine the impact on the node from one of six different columns: capacity threshold (disruption), inventory loss (disruption), demand variation (disruption), new design stock (response mechanism) or new availability and OEE (response mechanism). Depending on the type of disruption/response mechanism the user was creating, other columns would blackout to indicate the user data is not necessary on those cells. Finally, users could complete the rest of the blank cells or leave them empty, which defaulted to applying that disruption/response mechanism to all the items of the blank column within the node selected.

Additionally, some extra tabs were included. One was for gathering the location of all nodes in the supply chain to later include it in the visualization dashboard. Another one was for collecting the price of the components, to calculate the impact on revenue of the simulation results. And two more with the suggested additional capacity and TTR that suppliers filled out on the survey, to provide a reference for users when defining disruption scenarios.

Overall, the Input File reinforced the modular and universal features of the tool. It further incorporated the configurable feature by empowering users to test any combination of scenarios and response mechanisms, add as many nodes to the model as possible or desired, and apply the tool to other brands.

For the project in scope, the tool was also considered practical because it enabled reducing the order of magnitude of the BCP process length from days to hours[6].

From a practical standpoint, perhaps configuring the file for the first time would be time-consuming for a user, but afterward, the user would only need to focus on playing

[6]Minutes for running scenarios. For reference: 2.11 min/scenario (simulation + print time to spreadsheet), considering 30 total nodes, a 9.5y simulation length, using an M3 Mackbook Pro.

with the Disruption Parameters tab, and updating the initial inventories and expected demand. Even if structural changes were required, which occurred infrequently, the Reconfigure Supply Chain tab and the error-preventing mechanisms would accelerate such a process.

An opportunity to further improve the tool's practicality would have been connecting the file to the company's system to automatically update the inventory position. Although doing so was considered, it was not implemented due to time constraints.

Data Visualization

Creating a data visualization dashboard was the missing piece towards completing the tool framework depicted in Figure 4-2. Doing so would also provide the visual feature the tool lacked so far. The creation process was iterative and incorporated user feedback in between versions.

To allow users to visually perform a comprehensive analysis of the supply chain, two sets of tabs were created within the business intelligence software: analytical and informational.

Analytical tabs. These tabs focused on showing the impact of each disruption scenario on revenue, capacity, and inventory to augment BCP practices. The goal of this section of the dashboard was to provide new insights about the pmdiTech supply chain to the GSC&S and ESM members at AZ, while helping them understand the consequences of a disruption on the performance and resilience indicators they considered valuable during the Define Stage. The data source for these tabs was primarily the Solutions Spreadsheet of the simulation.

- A *Summary* tab, shown in Appendix C-21, was designed to guide users on their analysis. In this tab, users were able to filter per scenario and view both the revenue and the costs of goods sold for each product. The tab also provided a high-level overview of how stressed inventory and capacity were throughout the simulation at AZ and tier-one supplier sites by leveraging RAG[7] indicators.

[7] Red, Amber, Green.

The capacity indicator displayed green whenever the machine capacity utilization at the node was below an ideal utilization level, amber when utilization was above the ideal level, and red whenever a site was running at maximum capacity. The inventory indicator displayed green when the inventory level was at or above the design stock level, amber when inventory dropped up to 50% below the design, and red when the inventory level fell more than that.

- A *Capacity Utilization* tab, shown in Appendix C-22, was designed to contrast the percentage utilization of any node, at a process or even machine level, across scenarios. An area chart, for each scenario selected, displayed three metrics: the capacity available, the ideal utilization, and the actual utilization.

  The ideal utilization was the capacity utilization percentage at which a machine would need to run to fulfill the estimated demand under normal circumstances, as designed per the ESM team. Such a parameter was included only in the dashboard, not in the simulation; therefore, it allowed validating if the simulation was behaving in line with what users had planned.

- An *Asset Utilization* tab, shown in Appendix C-23, was designed to complement the Capacity Utilization tab. This tab followed the same structure, but the difference was that the charts were in units of material instead of percentages. Charts within this tab facilitated identifying the periods when additional capacity investments were required at a node for a certain process.

- An *Inventory Coverage Scenarios* tab, shown in Appendix C-24, was designed to display the evolution of the weeks of supply of a component held at a node. The area chart in this tab overlayed scenarios to understand the impact that each disruption scenario had on inventory. This tab would show users if a product ran out of inventory and when that would happen, in case it did.

- An *Inventory Coverage Components* tab, shown in Appendix C-25, was designed to complement the previous tab. Beyond comparing the effect of a disruption scenario on an individual component, overlaying the inventory coverage of multiple components within a node was also considered relevant.

- An *Inventory* tab, shown in Appendix C-26, was designed to mimic the Inventory Coverage Scenarios tab by displaying the chart in inventory units instead of weeks of supply. One of the assumptions of the model was that warehouses had unlimited capacity. Although in reality that is not true, such an assumption gave users perspective of the consequences of maintaining an inventory policy based on weeks of supply. The chart in this tab exposed the warehouse capacity required in the future to maintain such policies or the time by which complementary resilience preventive measures had to come into effect to cover the risk.

- A *Node Analysis* tab, shown in Appendix C-27, was designed to display both inventory and capacity charts at a node level. The inventory charts showed the inventory coverage of each component compared to the amber and red RAG indicator thresholds, while the capacity tabs tracked the capacity utilization and the capacity installed at a process level.

- Finally, a *Simulation Demand* tab, shown in Appendix C-28, was designed to let users visualize their forecasted demand per brand compared to the simulation demand and the on-time deliveries within the simulation. This tab supported users to analyze the impact on lost sales, particularly when the demand penalty condition was active in the model.

Informational tabs. These tabs focused on showcasing additional aspects of the brand. The objective of this section of the dashboard was to enable users to validate if the supply chain configuration they wanted to model was accurate. Beyond BCP, these tabs would be able to enrich conversations within the organization. The Input File was the primary source of data for these tabs.

- The *Brand* tab, shown in Appendix C-29, displayed a breakdown of the demand for the brand portfolio. Charts in this tab displayed the expected demand growth by product, the evolution of the portfolio composition by product, and the demand split by country.

- The *Supply Chain Network* tab, shown in Appendix C-30, used a Sankey chart to show the relationships between the nodes included in the model. This tab included a component filter, to visualize which nodes traded such components, and a node filter to identify who were the suppliers of the node in scope.

- The *Bill of Materials* tab, shown in Appendix C-31, used a Sankey chart to show the flow of components through processes across the value chain up to the brand's products. By filtering by nodes it was possible to visualize the connection of raw materials with processes and finished products within the node. By filtering by product it was possible to identify the components and processes required to manufacture the inhaler at the AZDP site.

- The *Supply Chain Map* tab, shown in Appendix C-32, showed the geographic location of the nodes represented in the model to identify clusters and potential regional risks. This display also allowed filtering the map by the external supplier categories described in Chapter 2.1.1.

- The *Installed Capacity* tab, shown in Appendix C-33, displayed the installed capacity over time. This tab provided a breakdown of the capacity by machine and product, according to the selected node. These visuals helped identify the supply chain bottlenecks and potential shifts in the bottlenecks over time.

- Finally, a *Disruption Parameters* tab, shown in Appendix C-34, was included to help users remember which were the parameters of each disruption scenario simulated.

To leverage chart interactions of business intelligence solutions, as those filters mentioned along the dashboard tabs description, relational structures between data sets were necessary. Those relationships were built within the software and facilitated the connectivity between the Input File and the Solutions Spreadsheet.

Overall, the dashboard created to analyze the supply chain simulation solutions provided the visual element the tool was missing. The dashboard views described above exemplify how business intelligence tools provide a broad range of possibilities for users to play with data, generate insights, and augment their BCP practices by further improving the tool's modular feature.

User Guide

An extra step to enhance the tool and improve its practicality was defining a user guide. This guide was built on a spreadsheet and had four tabs: (1) the instructions to set up the Back End of the tool; (2) the instructions to set up the Front End of the tool; (3) the instructions on how to run simulations; and (4) a set of Frequently Asked Questions.

Further explaining this guide would be out of scope for this thesis. Nevertheless, it was generally appreciated among stakeholders and it unveiled opportunities to improve the tool throughout the Prototype Stage.

In the end, the Prototype Stage proved how valuable the Ideate Stage was for conceiving a useful product. It also allowed identifying some of the tool's limitations objectively. It further demonstrated that the Design Thinking methodology truly is iterative.

4.6 Test Stage: Assessing Impact

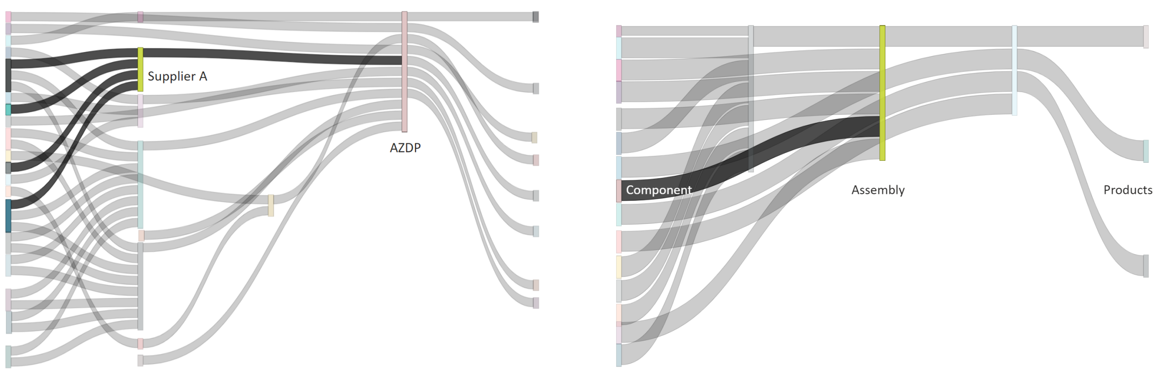
Testing sought to validate if the tool met the business needs. So far, the tool already served its first purpose; it was an in-house tool that allowed AZ to stress-test the supply chain resilience of their brands for defining BCP. The second purpose was to make the tool a viable building block for a supply chain digital twin. To achieve this, it had to prove that it satisfied a business case.

4.6.1 Business Case

Disclaimer: The business case evaluated in this thesis is based on actual events. However, to preserve the confidentiality of proprietary information and ensure the privacy of the organization, certain details have been modified, altered, or concealed, and sensitive data have been omitted.

Around the time the tool was being developed, the ESM team was having conversations with Supplier A to decide if it was worth investing in capacity at a second site. Supplier A, an AZDP tier-one supplier with multiple tier-two suppliers as highlighted in Figure 4-5a, manufactured a device component used in the assembly process at AZDP as highlighted in Figure 4-5b.

Filtering Figure 4-5a by component would illustrate all the unique nodes supplying the part to AZ; however, in this case, the chart was not included for simplicity because Supplier A was AZDP's sole provider for such a component and only manufactured it at one site.



(a) Suppliers Sankey (b) Components Sankey

Figure 4-5: Business case: context of supplier

Note: Goods flow towards the right of the charts

AZ had three machines at Supplier A's first site and had already planned on having a fourth machine up and running by 2027. Investing in capacity at Supplier A's second site was not necessary, but would serve as a backup mechanism in case of disruption - it would be redundant capacity. The company was further skeptical about the investment because the redundant capacity would be available from 2026 onwards and would only provide one-third of the main site's output.

Although financial considerations mattered, the decision driver was serving patients. Therefore, the main question was how much difference in on-time delivery would the investment in additional capacity make?

To evaluate the benefit of having the second site, in addition to having the fourth machine, two scenarios were simulated in addition to the baseline:

- One with a *fire* at Supplier A's main site on New Year's Eve 2025. Assuming that the *company did not invest* in the redundant capacity, that the finished goods *warehouse was not affected*, and that the TTR for Supplier A would have been 2 years, accounting for the site's reconstruction and the regulatory approval process. The scenario also considered that the *fourth machine was installed* at the second site.

- One with a *fire* at Supplier A's main site on New Year's Eve 2025. Assuming that the *company did invest* in the redundant capacity, that the finished goods *warehouse was not affected*, and that the TTR for Supplier A would have been 2 years, accounting for the site's reconstruction and the regulatory approval process. The scenario also considered that the *fourth machine was installed* at the second site.

To represent the problem in the Input File, as shown in Figure 4-6, the capacity at the supplier was equally divided into three machines:

- For the first scenario, 'Fire at Supplier A - no backup', the capacity for all three machines was limited to zero from Jan 2026-Jan 2028, and any WIP inventory was completely depleted.

- For the second scenario, 'Fire at Supplier A - with backup', only the capacity of two machines was limited to zero from Jan 2026-Jan 2028, and any WIP inventory was completely depleted.

  The third machine was not affected in this scenario because it represented the redundant capacity at the backup site. This assumed operational conditions would have been identical at the second site, compared to the first site, although those details would not be relevant unless the results suggested that shortages were avoided for only a few weeks or days under this scenario.

Those scenarios were then simulated with the tool and analyzed in the Power BI dashboard.

Run?	Scenario	Start Date	End Date	Node	Component	Process	Machine	Capacity Threshold (%)	Inventory Loss (%)	Demand Variation (%)	New Design Stock (WOS)	New Availability (days/week)	New OEE (%)
TRUE	Fire at Supplier A - no backup	Jan-26	Jan-28	Supplier A		Component process	Machine 1	0%					
TRUE	Fire at Supplier A - no backup	Jan-26	Jan-28	Supplier A		Component process	Machine 2	0%					
TRUE	Fire at Supplier A - no backup	Jan-26	Jan-28	Supplier A		Component process	Machine 3	0%					
TRUE	Fire at Supplier A - no backup	Jan-26		Supplier A		Component process			100%				
TRUE	Fire at Supplier A - with backup	Jan-26	Jan-28	Supplier A		Component process	Machine 1	0%					
TRUE	Fire at Supplier A - with backup	Jan-26	Jan-28	Supplier A		Component process	Machine 2	0%					
TRUE	Fire at Supplier A - with backup	Jan-26		Supplier A		Component process			100%				

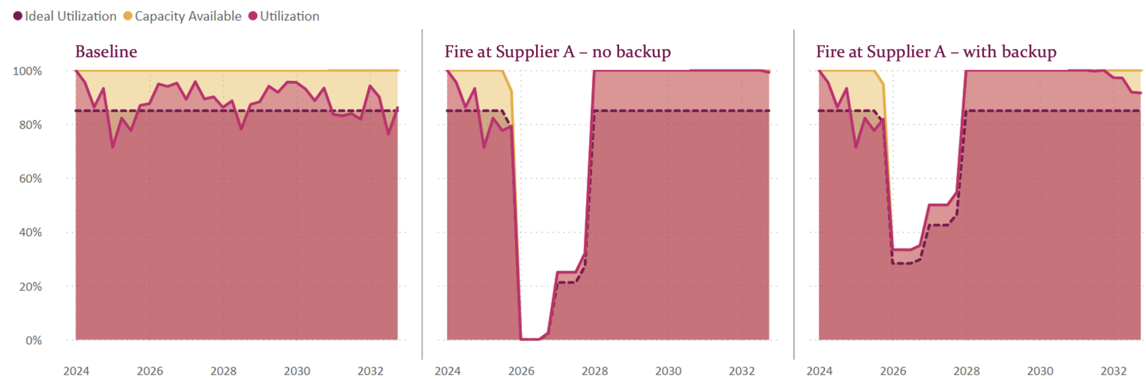
Figure 4-6: Business case: input file

From a capacity perspective, looking only at Supplier A was worrisome, as seen in Figure 4-7. The case without a backup machine showcased how nothing was produced during the first year of the disruption, how the fourth machine produced at maximum capacity during the second year of the disruption, and how Supplier A continued producing as much as possible for the rest of the simulation even after recovering from the fire. The case with a backup also suggested that capacity at Supplier A would have remained stressed despite having relied on the redundant machine during the two years the main site was offline.

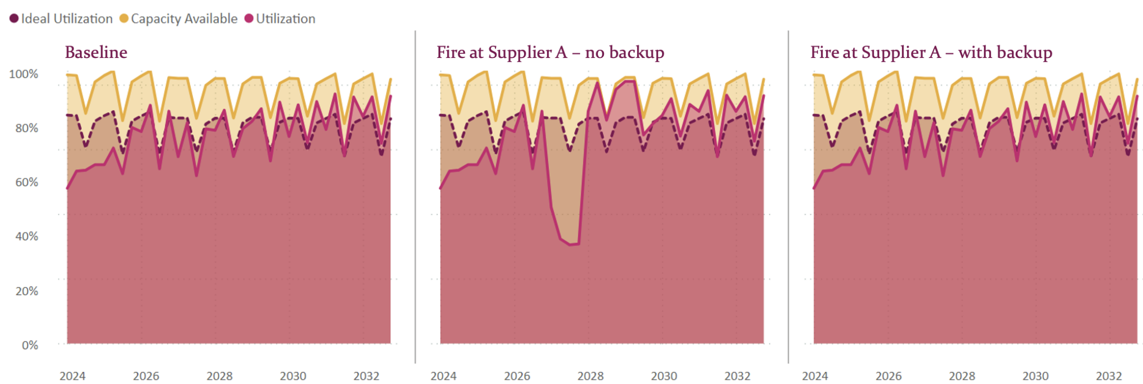
At the company's site, the story was different. In the case with no backup, the capacity utilization fell to nearly half the level of the baseline although there was capacity available, implying that a limited supply of the component manufactured by Supplier A was capping the production. After Supplier A recovered, the company's utilization jumped almost to the maximum levels for more than a year before normalizing. Meanwhile, the scenario with a backup machine suggested there was no impact when compared to the baseline.

To validate the insights from the capacity charts, a similar analysis was also conducted on the inventory coverage charts in Figure 4-8 by following the path from Supplier A to the consumer. Doing so would demonstrate the time-to-survive in case of stockout.

Supplier A's inventory coverage chart displayed how in the scenario without a backup, the device component's inventory depleted after four months, while the same inventory lasted nearly a quarter longer in the scenario with a backup. At the company's site, the depletion took longer because they held a larger amount



(a) Supplier A



(b) Company Site

Figure 4-7: Business case: quarterly capacity utilization

of inventory. In the scenario without a backup, the company had used all their component's inventory by 2026 and ran out of finished goods inventory for Product X approximately a semester later; they stocked out despite leveraging the capacity from the fourth machine during 2027. Meanwhile, in the scenario with a backup, the production of the redundant machine helped the company resume their operations as normal, even without affecting the finished goods inventory position.

All these inventory charts also helped explain why utilization at Supplier A's site never returned to the ideal levels. In the case with a backup machine, it took the company more than five years to get back to their designed stock level, while in the case without a machine the inventory reached the baseline level until the end of the simulation period. In turn, Supplier A was never able to return to the baseline because of the ripple effect from the disruption. Such behavior illustrated the concept of time-away-from-design that the company was interested in analyzing.

So far the analysis depended on the assumption that the fire would not have impacted the warehouse and that the fourth machine would fit at the supplier's secondary site. Leveraging the tool, it was possible to test these assumptions and further enhance the analysis by evaluating four more scenarios:

- One with a *fire* at Supplier A's main site on New Year's Eve 2025. Assuming that the *company did not invest* in the redundant capacity, that the finished goods *warehouse was affected*, and that the TTR for Supplier A would have been 2 years, accounting for the site's reconstruction and the regulatory approval process. The scenario also considered that the *fourth machine was installed* at the second site.

- One with a *fire* at Supplier A's main site on New Year's Eve 2025. Assuming that the *company did invest* in the redundant capacity, that the finished goods *warehouse was affected*, and that the TTR for Supplier A would have been 2 years, accounting for the site's reconstruction and the regulatory approval process. The scenario also considered that the *fourth machine was installed* at the second site.

- One with a *fire* at Supplier A's main site on New Year's Eve 2025. Assuming that the *company did not invest* in the redundant capacity, that the finished goods *warehouse was affected*, and that the TTR for Supplier A would have been 2 years, accounting for the site's reconstruction and the regulatory approval process. The scenario also considered that the *fourth machine was not installed* at the second site.

- One with a *fire* at Supplier A's main site on New Year's Eve 2025. Assuming that the *company did invest* in the redundant capacity, that the finished goods *warehouse was affected*, and that the TTR for Supplier A would have been 2 years, accounting for the site's reconstruction and the regulatory approval process. The scenario also considered that the *fourth machine was not installed* at the second site.

Beyond similarly analyzing these scenarios as the previous two, it was important to compare the impact of each on the on-time deliveries. Visually, a chart overlaying the inventory coverage of the products manufactured by the company helped ratify that scenarios without a backup machine performed worse and that removing the assumptions from the analysis the consequences worsened. Figure 4-9 shows such charts across each set of scenarios.

Visual insights created awareness, provided a strong storyline behind each scenario and showcased the tool's versatility. Nevertheless, linking the impact to performance metric insights was crucial to garner support from key stakeholders.

To answer how much difference in on-time delivery would the investment in additional capacity make, data from each simulated scenario was presented in a table comparing the percentage of products delivered on time to patients with respect to the baseline scenario.

The results in Table 4.4 demonstrated how investing in backup capacity could substantially increase the percentage of timely deliveries within the context of this case example. The average value for on-time deliveries across all scenarios with a redundant machine was 97.5% compared to 90.2% in all the cases when the fourth machine was installed at the second site, 69% when no fourth machine was available, and 61.7% when no backup was available.

	No 4th machine	With 4th machine	Difference
No backup	70.5%	88.6%	18.1%
With backup	98.9%	100.0%	1.1%
Difference	28.4%	11.4%	

(a) Scenarios without warehouse impact at Supplier A's main site

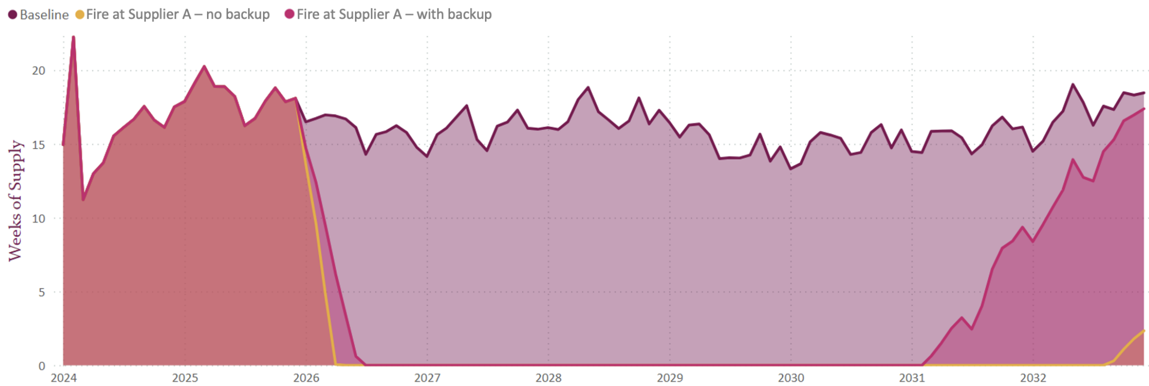
	No 4th machine	With 4th machine	Difference
No backup	14.8%	72.9%	58.0%
With backup	91.9%	99.4%	7.4%
Difference	77.1%	26.5%	

(b) Scenarios with warehouse impact at Supplier A's main site

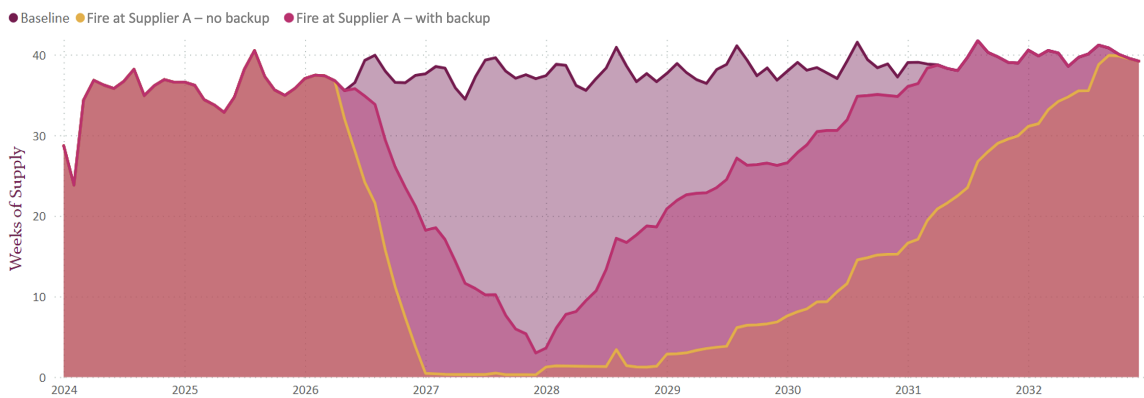
Table 4.4: Business case: on-time deliveries results

Regardless of the conclusions of this particular example, overall, the case presented above proved valuable to stakeholders given the tool's potential to augment BCP practices across the organization. Although the performance and resilience metrics were not explicitly stated in the tool, the case showcased how the business intelligence dashboard was well designed to effectively understand the impact of disruptions and support decision-making.

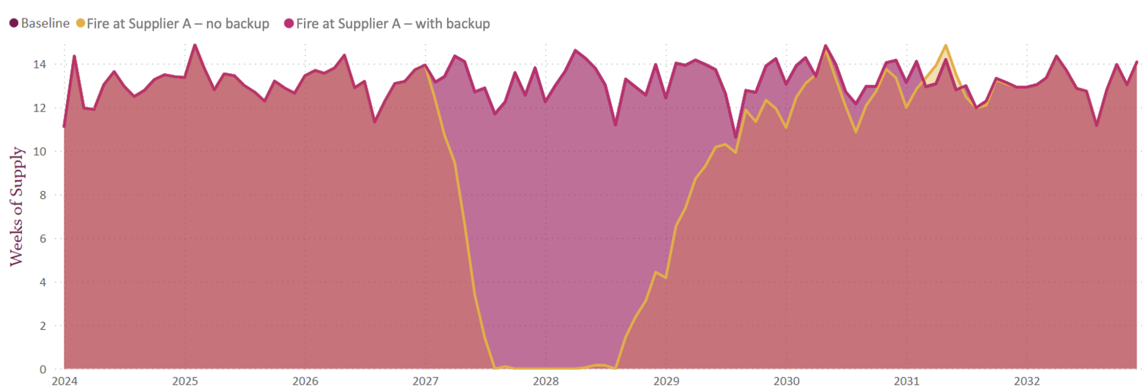
The Test Stage demonstrated the importance of presenting the tool's capabilities in a relevant business context and with well-founded arguments. It demonstrated how the tool matched users and business needs, while showcasing the potential benefits and applications of scaling this digital development. It further validated the achievement of the thesis project's goals.



(a) Supplier A - component finished good

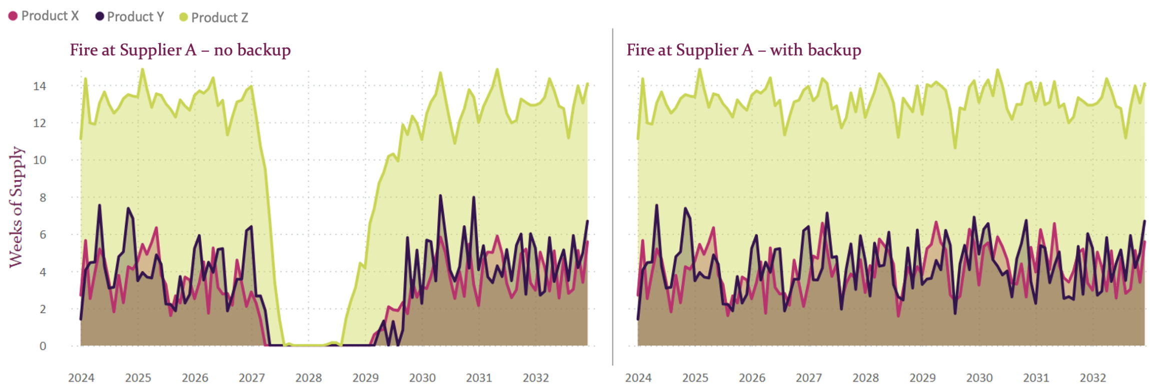


(b) Company Site - component raw material

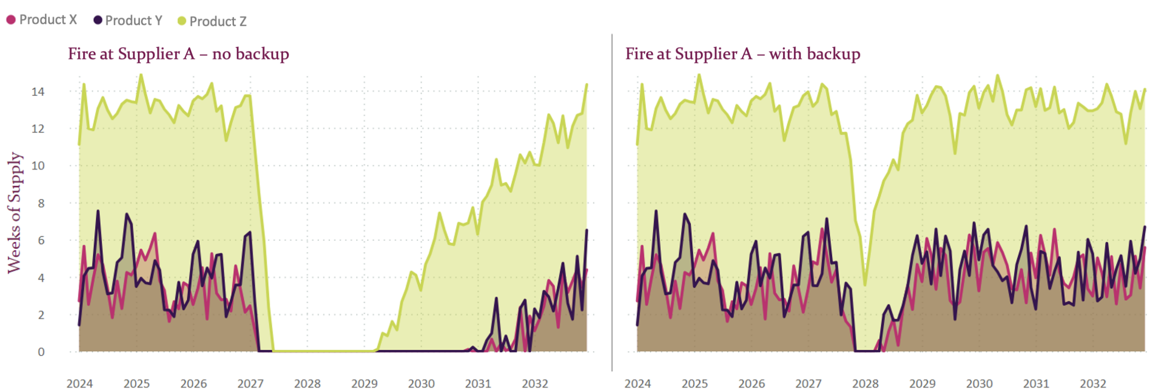


(c) Company Site - product finished good

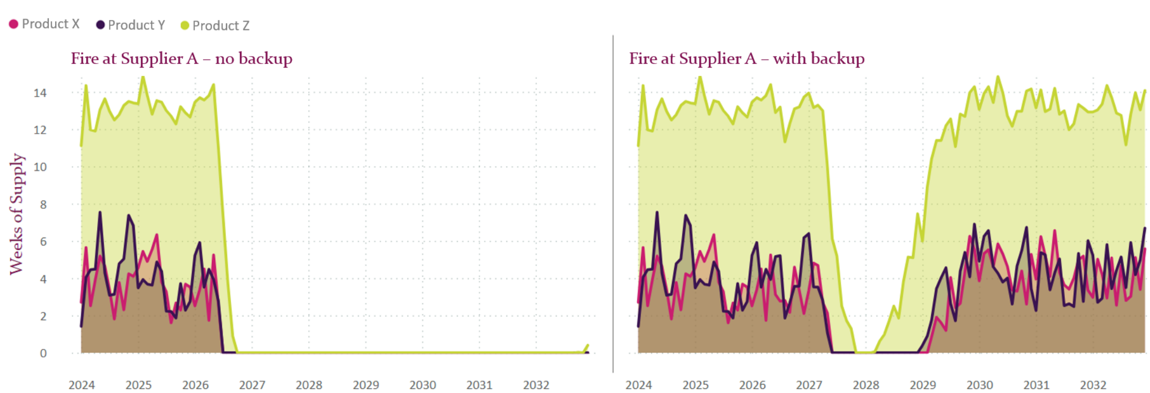
Figure 4-8: Business case: inventory coverage by node



(a) Scenarios with fourth machine and without warehouse impact at Supplier A's main site



(b) Scenarios with fourth machine and with warehouse impact at Supplier A's main site



(c) Scenarios without fourth machine and with warehouse impact at Supplier A's main site

Figure 4-9: Business case: inventory coverage by set of scenarios

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 5

Results

The tool presented in this thesis was designed to enable AZ to conduct stress tests and define BCPs for their brands, laying the groundwork for developing a supply chain DT. The tool's alignment with the Define Stage scope agreements was evident in several aspects:

- The configuration of the tool allowed users to test all the desired disruption scenarios and it also included response mechanisms to test potential BCP policies.
- Although the performance and resilience metrics were not explicitly included in the tool's dashboard, the tabs designed allowed users to inspect the desired metrics visually.
- Besides including the supply chain nodes between the filling and packing stages, this version of the tool also captured the demand at a country level and included some nodes upstream in the API and Excipients stage.
- The tool encompassed both enabling technologies, simulation and business intelligence, ensuring the compatibility for further development toward DT status.

Beyond enabling AZ to outline BCPs more effectively, the tool provides additional use cases. In case of disruption, it would allow AZ to understand the extent of the potential impact and respond swiftly. When launching medicines, AZ would be able to better design resilient supply chains since their inception and validate that the supply chain design is robust enough to meet the expected demand. Similarly, the tool would be able to support planning for equipment investments or divestments. More importantly, when working with suppliers, it would facilitate collaboration towards resilience.

While the tool achieved its primary objectives, it presents several enhancement opportunities due to constraints in time, data, and budget. Future iterations could connect the tool to the company's databases for streamlining data updates, extend the simulation script to encompass TTS calculations for all nodes or other optimization models, integrate flexibility-based resilience mechanisms alongside existing redundancy mechanisms, and even incorporate sustainability metrics.

Nevertheless, the resilience features embedded in the tool provide a solid foundation to expand the tools' use to other resilience cases beyond disruption testing, depending on the experience and maturity of the organization.

Additionally, beyond the objectives of the thesis project, the approach to the problem is generalizable to any organization seeking to improve its supply chain resilience practices through the use of digital developments.

5.1 Supply Chain Resilience Tool Building Framework

The methodology chapter detailed the systematic approach employed to develop and test the supply chain resilience tool.

The process began with extensive stakeholder interviews to gather insights into the challenges and needs related to supply chain resilience, ensuring the tool addressed real-world problems and user requirements.

Subsequently, a more detailed project's scope was defined along with the desired tool's characteristics and functionalities, leading to a blueprint of the data required for developing the tool.

Data collection followed, focusing on gathering information from internal and external sources. Meetings with suppliers, defining questionnaires, and reviewing contracts were unveiled as a means to fill data gaps.

In parallel, creative solutions were brainstormed, and a conceptual framework for the tool was developed. Assumptions and boundaries were established to create a model that was both robust and flexible, capable of simulating a variety of supply chain scenarios.

The process continued with the development of a Front End and a Back End for the tool.

- In this case, the Back End leveraged discrete event simulation for its effectiveness in modeling complex supply chain dynamics over time. This simulation was coded with an emphasis on modularity and reusability, ensuring the tool could be adapted for various scenarios.

- For this project, the Front End was divided into two. First defining a user interface design, focusing on creating an intuitive and error-resistant experience through dynamic dropdown lists, data validation, and protective measures against user input errors. Then, constructing a comprehensive data visualization dashboard within business intelligence software, enabling users to analyze supply chain performance visually. These portions of the tool were developed with an emphasis on configurability and practicality, within the constraints of the project.

Finally, the tool was tested in a business context, showcasing the potential use cases for enhancing resilience practices within the organization.

The design thinking methodology applied to supply chain resilience underpinned the tool's development, ensuring it was robust, user-friendly, and capable of providing meaningful insights into supply chain resilience.

Figure 5-1 illustrates a general framework for applying the methodology to digital developments in supply chain resilience.

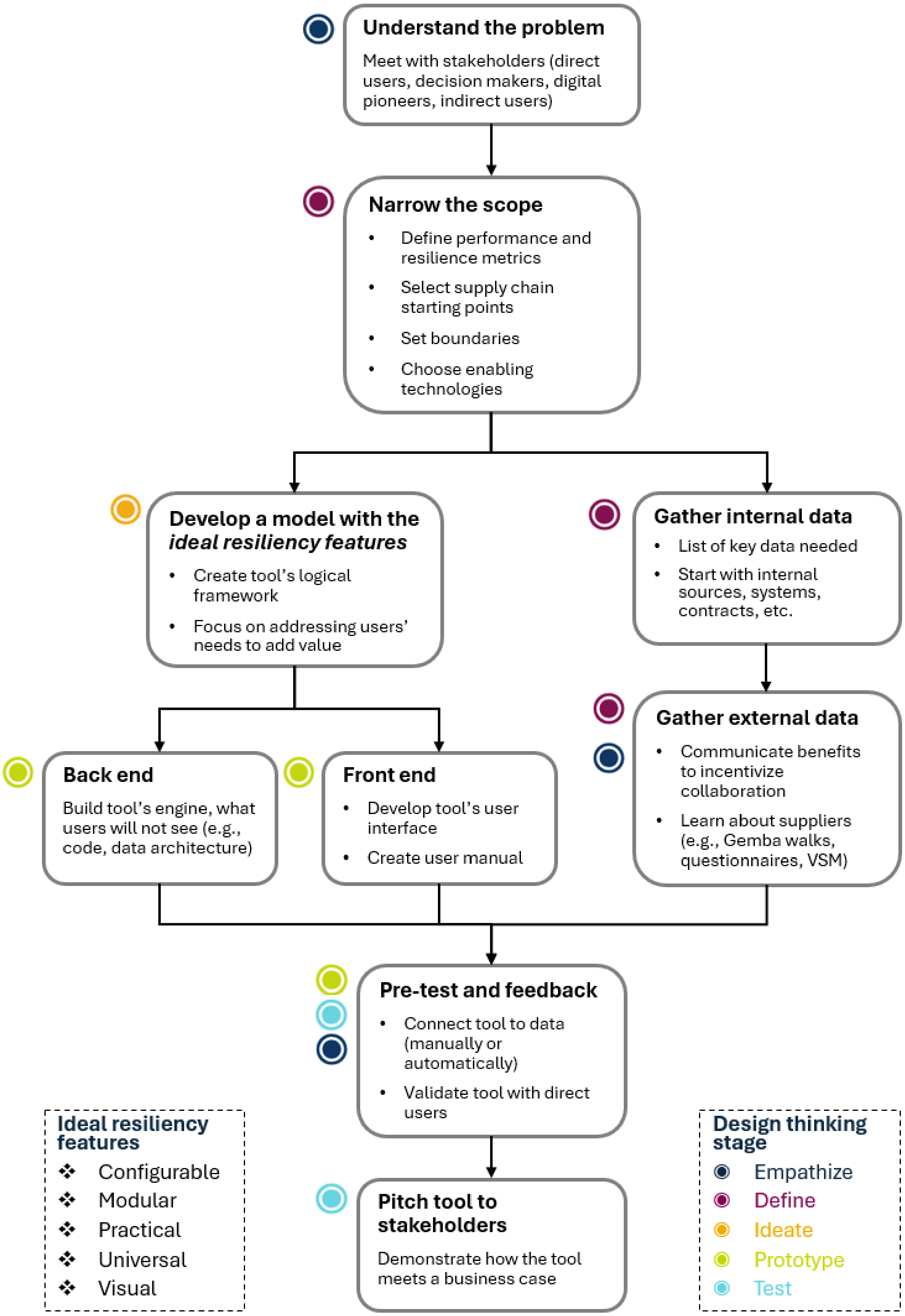


Figure 5-1: General framework for developing supply chain resilience digital tools

5.2 Supply Chain Digital Twin Transition

The solution presented for this thesis project leveraged simulation and visualization capabilities. However, the tool developed was limited to visual diagnostic analysis.

The tool virtually represented the physical supply chain and encompassed some enabling technologies to analyze data, but the tool only achieved a digital model status given the manual interactions between the physical and the digital entities.

For organizations aspiring to upgrade their digital models, a robust data infrastructure is essential because continuous access to data opens the possibility of incorporating predictive and prescriptive analytical capabilities through machine learning and artificial intelligence.

To transition from a static digital model to a dynamic digital shadow, the natural step forward would involve investing in sensors to collect data and cloud computing to update the digital tool automatically. However, doing so in a supply chain context requires data sharing between organizations because a single entity rarely constitutes an entire supply chain.

A valuable lesson while gathering data was that by communicating the long-term advantages of DTs in supply chain contexts, discussions evolved from data ownership concerns to proactive collaboration on data integration. Conversations shifted from 'Why do you want our data?' to 'How can we streamline data connections between our facilities?'.

Fostering a collaborative mindset unlocks the potential to develop supply chain digital shadows, and organizations that successfully develop digital shadows stand to gain competitive advantages through enhanced decision-making capabilities. More importantly, collaboration also enhances visibility, which is crucial for augmenting resiliency.

For organizations aspiring to further upgrade their digital shadows, implementing the necessary technology to modify the physical supply chain should prove time-consuming but not an impediment to achieving digital twin status. However, since the essence of a DT is to enable automatic enhancement between physical and digital entities in the twin, the challenge will be facilitating the unsupervised interaction between these entities because supply chains are generally not vertically integrated.

- To overcome the challenge, most organizations will opt for a practical approach: developing individually owned DTs that address the supply chain priorities of the organization. These DTs shall make decisions based on what they consider optimal for a single supply chain entity and affect only the physical portion of the supply chain controlled by that entity, while leveraging data from the multiple supply chain entities.

  This solution implies that entities in the supply chain still collaborate to collect and share data, but each entity develops its own DT for specific supply chain applications, such as inventory management or production planning. Thus, *organizations following this approach will have a 'digital twin for supply chain applications'.*

- Alternatively, some organizations may try to follow a more holistic approach: developing a single DT that makes decisions based on what it considers optimal for the entire supply chain, affecting any physical portion of the supply chain regardless of whom it belongs to.

  This solution implies that entities in the supply chain align incentives and agree upon the underlying metrics for which the DT optimizes, particularly because the decisions made by the DT will usually involve cost tradeoffs. Thus, *following the holistic approach should prove difficult, but organizations that successfully do so will have a true 'supply chain digital twin'.*

The main difference between the two approaches organizations may follow when upgrading their digital shadows to digital twins is that the practical approach leads to having a DT that optimizes locally ('digital twin for supply chain applications'), while the holistic approach leads to having a DT that optimizes globally ('supply chain digital twin'). Although both strategies should drive value for organizations, aspiring for a 'supply chain digital twin' arguably unlocks the potential to generate larger benefits.

Chapter 6

Conclusions

In conclusion, the thesis documented the successful development of a digital tool aimed at bolstering supply chain resilience through simulation and business intelligence capabilities. The tool - characterized by its configurability, modularity, practicality, universality, and visuality - enabled supply chain managers at AZ to proactively manage disruptions. By integrating design thinking and a thorough methodology that included stakeholder interviews, data collection, and creative problem-solving, the project yielded a robust and flexible tool that not only fulfilled the initial objectives but also offered broader applications for supply chain management.

The tool was tested in real-world business applications, illustrating its capacity to significantly enhance BCP processes, facilitate swift responses to disruptions, and assist in the strategic planning of supply chain designs and investments. It addressed the dynamic nature of supply chains by allowing for continuous revisions and updates to the BCPs as nodes and relationships evolve. It also aided in navigating path-dependent decisions by simulating various disruption scenarios, thus informing strategic choices that consider both present and future risks. Moreover, the tool enhanced visibility across supply chain tiers, facilitating collaboration with external stakeholders and enabling a more comprehensive understanding of the supply chain network.

Beyond AZ, the methodologies and principles applied in this project are indicative of their potential for widespread adoption across various organizations, signaling a move towards a more resilient and digitally-enabled supply chain management approach. Consequently, this thesis also provided a framework for developing digital tools that foster supply chain resilience.

The digitalization of supply chain management is advancing, and this project demonstrated that the journey from a digital model to a digital shadow, and ultimately to a digital twin, is a transformative process that requires not only technological innovation but also a cultural shift towards collaboration and data sharing. Organizations that embrace these changes are poised to reap the benefits of enhanced agility, efficiency, and competitive advantage in an increasingly complex global market.

While this thesis has laid a solid foundation for adopting resiliency through digital models, future research could explore incentive alignment models to harmonize stakeholder objectives across the supply chain with aims at impulsing the development of supply chain digital twins that operate holistically at a network level instead of at a node level. In the end, seeking supply chain resilience should be a collaborative effort, not an individual one.

Appendix A

Company Appendix

A.1 Brand Details

Product	Total 2022	Total 2023
Device A1	44	46
Device A2	45	73
Device A3	-	1

Table A.1: pmdiTech Global Presence[1] [1][23]

[1] number of countries where the product has been approved

Product	2016	2017	2018	2019	2020	2021	2022	2023
Device A1	2	16	33	42	48	54	58	58
Device A2	-	-	-	2	28	203	398	677
Device A3[3]	-	-	-	-	-	-	-	-

Table A.2: pmdiTech Reported Sales by Product and Year [1] [2]

[1] in millions of dollars

[2] data from AZ Annual Reports 2016-2023 [48] [49] [50] [51] [24] [52] [23] [1]

[3] launched until 2024

A.2 Product Details

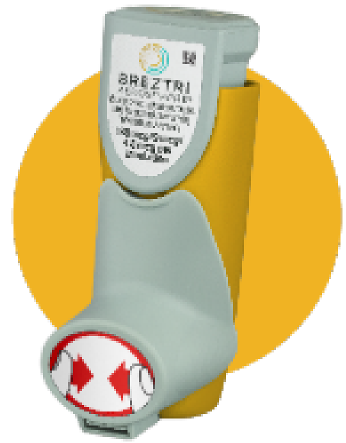
Product	API	Starting Material	Propellant
Device A1	Glycopyrrolate Formoterol fumarate	DSPC [1,2-Distearoyl-sn-glycero3-phosphocholine] Calcium chloride	Hydrofluoroalkane (HFA 134a)
Device A2	Budesonide Glycopyrrolate Formoterol fumarate	DSPC [1,2-Distearoyl-sn-glycero3-phosphocholine] Calcium chloride	Hydrofluoroalkane (HFA 134a)
Device A3	Albuterol sulfate Budesonide	DSPC [1,2-Distearoyl-sn-glycero3-phosphocholine] Calcium chloride	Hydrofluoroalkane (HFA 134a)

Table A.3: Product Components [53] [54] [55]

(a) Device A1 [56]   (b) Device A2 [57]   (c) Device A3 [58]

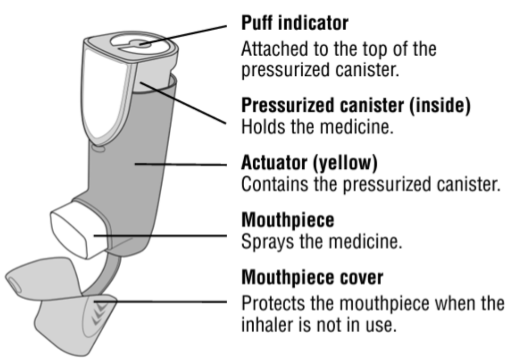
Figure A-1: pmdiTech Portfolio

(a) Original inhaler

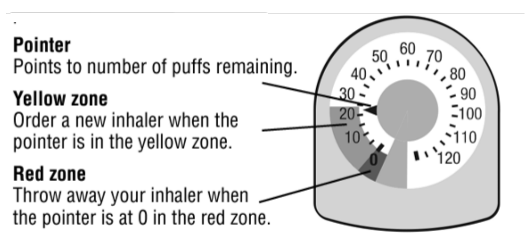


(b) New look inhaler

Figure A-2: Device A2 models comparison [57]



(a) Front View



(b) Top View

Figure A-3: Parts of Device A2 inhaler [59]

THIS PAGE INTENTIONALLY LEFT BLANK

Appendix B

Back End Appendix

B.1 Code

```
# %% [markdown]
# # Libraries

# %%
import pandas as pd
import math
import uuid
import datetime
import warnings

# %%
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=UserWarning)

# %% [markdown]
# # Objects

# %% [markdown]
# ## Node

# %%
```

```
class Node: # supply chain node
    def __init__(self, node_name):
        self.name = node_name
        self.raw_materials = {}
        self.finished_goods = {}
        self.bill_of_materials = {}
        self.holidays = []
        self.processes = {} # add in order!
        self.process_sequence = {}
        self.orders = {'received':[], 'wip':[], 'completed':[], '
    shipped':[], 'delivered':[], 'placed':[]}

    def addOrder(self, order, type):
        self.orders[type].append(order)

    def removeOrder(self, order, type):
        self.orders[type].remove(order)

    def addRawMaterial(self, product):
        self.raw_materials[product.name] = product

    def addFinishedGoods(self, product):
        self.finished_goods[product.name] = product
        self.process_sequence[product.name] = []
        self.bill_of_materials[product.name] = {}

    def addProcess(self, process):
        self.processes[process.name] = process

    def addProcessSequence(self, product_name, process_sequence_list
    ):
        self.process_sequence[product_name] += process_sequence_list

    # component = required component (product or process); process =
     process where the component is required
    def addBOM(self, finished_good_name, component_name, quantity,
```

```
process_name):
        if component_name not in self.bill_of_materials[finished_good_name].keys():
            self.bill_of_materials[finished_good_name][component_name] = {process_name: quantity}
        else:
            self.bill_of_materials[finished_good_name][component_name][process_name] = quantity

# %% [markdown]
# ## Process

# %%
class Process: # manufacturing process
    def __init__(self, process_name, initial_inventory=0, design_stock=10, generic=False):
        self.name = process_name
        self.inventory = [initial_inventory]
        self.design_stock = [design_stock]
        self.inventory_gap = design_stock - initial_inventory
        self.annual_demand = 0
        self.time_to_survive = [design_stock]
        self.machines = {}
        self.generic = generic
        self.orders = []
        self.capacity_available = 0

        if generic:
            self.addMachine('generic')

    def addMachine(self, machine_name, capacity=1e10, availability=7, oee=1.0, start_date=0, end_date=624):
        self.machines[machine_name] = {}
        self.machines[machine_name]['production'] = [0]
        self.machines[machine_name]['utilization'] = [0]
        self.machines[machine_name]['initial_availability'] =
```

```
availability
        self.machines[machine_name]['initial_oee'] = oee
        self.machines[machine_name]['capacity'] = capacity
        self.machines[machine_name]['availability'] = availability
        self.machines[machine_name]['oee'] = oee
        self.machines[machine_name]['capacity_available'] = [math.
floor(capacity*availability*oee)]
        self.machines[machine_name]['percent_available'] = [1]
        self.machines[machine_name]['start_date'] = start_date
        self.machines[machine_name]['end_date'] = end_date
        self.machines[machine_name]['capacity_threshold'] = 1
        self.capacity_available += math.floor(capacity*availability*
oee)

    def addOrder(self, order):
        self.orders.append(order)

    def removeOrder(self, order):
        self.orders.remove(order)

# %% [markdown]
# ## Product

# %%
class Product: # finished goods/raw material
    def __init__(self, item_name, initial_inventory=0, design_stock
=10):
        self.name = item_name
        self.inventory = [initial_inventory]
        self.design_stock = [design_stock]
        self.inventory_gap = design_stock - initial_inventory
        self.annual_demand = 0
        self.time_to_survive = [design_stock]
        self.late_delivery = 0
        self.replenishment_time = 0
        self.suppliers = {}
```

```
        self.customers = {}

    def addSupplier(self, supplier, minimum_order_quantity=1, order_lead_time=6, accelerated_lead_time=5):
        self.suppliers[supplier.name] = (supplier, minimum_order_quantity, order_lead_time, accelerated_lead_time)
        self.replenishment_time = max(self.replenishment_time, order_lead_time)

    def addCustomer(self, customer, delivery_lead_time=1):
        self.customers[customer.name] = (customer, delivery_lead_time)

# %% [markdown]
# ## Order

# %%
class Order: # purchase/production order
    def __init__(self, id, date_received, supplier, customer, product, demand, due_date, internal=False, process=None):
        self.id = id
        self.status = 'received'
        self.internal = internal
        self.date_received = date_received
        self.supplier = supplier
        self.customer = customer
        self.product = product
        self.demand = demand
        self.date_due = due_date
        self.process = process
        self.amount_produced = 0
        self.date_production_start = 0
        self.process_amounts_required = []
        self.date_production_end = 0
        self.amount_shipped = []
        self.date_shipped = []
```

```
        self.amount_delivered = []
        self.date_delivered = []
        self.date_fulfilled = 0

# %% [markdown]
# ## Disruption

# %%
class Disruption:
    def __init__(self, date_start=0, date_end=1000, node_name='',
    product_name='', process_name='', machine_name='',
    capacity_threshold=1.0, inventory_loss=0.0, demand_variation=0.0,
     design_stock=0, new_availability=7, new_oee=0.60, type=''):
        self.date_start = date_start
        self.date_end = date_end
        self.node_name = node_name
        self.product_name = product_name
        self.process_name = process_name
        self.machine_name = machine_name
        self.capacity_threshold = capacity_threshold
        self.inventory_loss = inventory_loss
        self.design_stock = design_stock
        self.demand_variation = demand_variation
        self.machine_availability = new_availability
        self.machine_oee = new_oee
        self.type = type

# %% [markdown]
# # Functions

# %% [markdown]
# ## Initialize Objects

# %%
def initSim(sim_duration, nodes_dict):
    '''
```

```
    Expands lists within node according to the simulation duration.

    Parameters
    ----------
    sim_duration : int
            The number of weeks the simulation will run.
    nodes_dict : dict
            Dictionary with all Nodes (class) involved in the simulation.
    '''

    for node in nodes_dict.values():
        for p_key, process in node.processes.items():
            process.inventory += [0]*(sim_duration)
            process.design_stock += [process.design_stock[0]]*(sim_duration)
            process.time_to_survive += [0]*(sim_duration)
            for m_key, machine in process.machines.items():
                machine['capacity_available'] += [0]*(sim_duration)
                machine['production'] += [0]*(sim_duration)
                machine['utilization'] += [0]*(sim_duration)
                machine['percent_available'] += [0]*(sim_duration)
        for key, raw_material in node.raw_materials.items():
            raw_material.inventory += [0]*(sim_duration)
            raw_material.design_stock += [raw_material.design_stock[0]]*(sim_duration)
            raw_material.time_to_survive += [0]*(sim_duration)
        for key, finished_good in node.finished_goods.items():
            finished_good.inventory += [0]*(sim_duration)
            finished_good.design_stock += [finished_good.design_stock[0]]*(sim_duration)
            finished_good.time_to_survive += [0]*(sim_duration)

    return

# %% [markdown]
```

```python
# ## Reset Objects

# %%
def resetSim(sim_duration, nodes_dict, demand_distortion):
    '''
    Clear orders at the nodes.
    Set inventories and machine capacity utilization / production
    back equal to zero.

    Parameters
    ----------
    sim_duration : int
            The number of weeks the simulation will run.
    nodes_dict : dict
            Dictionary with all Nodes (class) involved in the
    simulation.
    '''

    for node in nodes_dict.values():
        for o_key, order_list in node.orders.items():
            order_list.clear()
        for p_key, process in node.processes.items():
            process.orders.clear()
            process.inventory[1:] = [0]*(sim_duration)
            process.design_stock[1:] = [process.design_stock[0]]*(
    sim_duration)
            process.time_to_survive[1:] = [0]*(sim_duration)
            process.annual_demand = 0
            for m_key, machine in process.machines.items():
                machine['capacity_available'][1:] = [0]*(
    sim_duration)
                machine['production'][1:] = [0]*(sim_duration)
                machine['utilization'][1:] = [0]*(sim_duration)
                machine['percent_available'][1:] = [0]*(sim_duration
    )
                machine['availability'] = machine['
```

```
initial_availability']
                machine['oee'] = machine['initial_oee']
                machine['capacity_threshold'] = 1
        for rm_key, raw_material in node.raw_materials.items():
            raw_material.inventory[1:] = [0]*(sim_duration)
            raw_material.design_stock[1:] = [raw_material.
design_stock[0]]*(sim_duration)
            raw_material.time_to_survive[1:] = [0]*(sim_duration)
            raw_material.late_delivery = 0
            raw_material.annual_demand = 0
        for fg_key, finished_good in node.finished_goods.items():
            finished_good.inventory[1:] = [0]*(sim_duration)
            finished_good.design_stock[1:] = [finished_good.
design_stock[0]]*(sim_duration)
            finished_good.time_to_survive[1:] = [0]*(sim_duration)
            finished_good.late_delivery = 0
            finished_good.annual_demand = 0

    demand_distortion = [1]*len(demand_distortion)

    return demand_distortion

# %% [markdown]
# ## Record Results

# %%
def recordSim(simulation_scenario, sim_duration, nodes_dict, dates,
    inventory_df, production_df, order_df):
    '''
    Captures inventories and production of the simulated scenario to
    its corresponding output dataframe.

    Parameters
    ----------
    simulation_scenario : str
            The name of the simulation scenario.
```

```
    sim_duration : int
            The number of weeks the simulation will run.
    nodes_dict : dict
            Dictionary with all Nodes (class) involved in the simulation.
    dates : list
            List with the dates of each simulation period.
    inventory_df : dataframe
            Pandas Dataframe where the simulation inventory indicators will be recorded.
    production_df : dataframe
            Pandas Dataframe where the simulation production indicators will be recorded.
    order_df : dataframe
            Pandas Dataframe where the simulation orders indicators will be recorded.
    disruption_df : dataframe
            Pandas Dataframe where the simulation disruption indicators will be recorded.

    Returns
    -------
    Updated inventory, production, order and summary frameworks.
    '''

    for n_key, node in nodes_dict.items():
        for p_key, process in node.processes.items():
            output_df = pd.DataFrame()
            output_df['Date'] = dates
            output_df['Scenario'] = simulation_scenario
            output_df['Node'] = n_key
            output_df['Type'] = 'Process'
            output_df['Component'] = p_key
            output_df['Design Stock'] = process.design_stock[1:]
            output_df['Inventory'] = process.inventory[1:]
            output_df['Time to Survive'] = process.time_to_survive
```

```
[1:]

        inventory_df = pd.concat([inventory_df, output_df])

    for rm_key, raw_material in node.raw_materials.items():
        output_df = pd.DataFrame()
        output_df['Date'] = dates
        output_df['Scenario'] = simulation_scenario
        output_df['Node'] = n_key
        output_df['Type'] = 'Raw Material'
        output_df['Component'] = rm_key
        output_df['Design Stock'] = raw_material.design_stock[1:]
        output_df['Inventory'] = raw_material.inventory[1:]
        output_df['Time to Survive'] = raw_material.time_to_survive[1:]

        inventory_df = pd.concat([inventory_df, output_df])

    for fg_key, finished_good in node.finished_goods.items():
        output_df = pd.DataFrame()
        output_df['Date'] = dates
        output_df['Scenario'] = simulation_scenario
        output_df['Node'] = n_key
        output_df['Type'] = 'Finished Good'
        output_df['Component'] = fg_key
        output_df['Design Stock'] = finished_good.design_stock[1:]
        output_df['Inventory'] = finished_good.inventory[1:]
        output_df['Time to Survive'] = finished_good.time_to_survive[1:]

        inventory_df = pd.concat([inventory_df, output_df])

    for p_key, process in node.processes.items():
        for m_key, machine in process.machines.items():
```

```
                output_df = pd.DataFrame()
                output_df['Date'] = dates
                output_df['Scenario'] = simulation_scenario
                output_df['Node'] = n_key
                output_df['Process'] = p_key
                output_df['Machine'] = m_key
                output_df['Utilization'] = machine['production'][1:]
                output_df['Utilization %'] = machine['utilization'][1:]
                output_df['Capacity Available'] = machine['capacity_available'][1:]
                output_df['Capacity Available %'] = machine['percent_available'][1:]
                output_df['Machine Start'] = dates[0] + pd.DateOffset(weeks=machine['start_date'])
                output_df['Machine End'] = dates[0] + pd.DateOffset(weeks=machine['end_date'])

                production_df = pd.concat([production_df, output_df])

        for order in node.orders['placed']:
            output_df = pd.DataFrame()
            output_df['Scenario'] = [simulation_scenario]
            output_df['ID'] = [order.id]
            output_df['Node'] = [order.customer.name]
            output_df['Supplier'] = [order.supplier.name]
            output_df['Component'] = [order.product.name]
            output_df['Internal'] = [order.internal]
            output_df['Demand'] = [order.demand]
            output_df['Production'] = [order.amount_produced]
            output_df['Date Placed'] = [dates[0] + pd.DateOffset(weeks=order.date_received-1)]
            output_df['Date Production Started'] = [dates[0] + pd.DateOffset(weeks=order.date_production_start-1)]
            if order.date_production_end >= order.
```

```
date_production_start:
            output_df['Date Production Ended'] = [dates[0] + pd.DateOffset(weeks=order.date_production_end-1)]
        else:
            output_df['Date Production Ended'] = [math.nan]
        if order.date_fulfilled >= order.date_received:
            output_df['Date Fulfilled'] = [dates[0] + pd.DateOffset(weeks=order.date_fulfilled-1)]
        else:
            output_df['Date Fulfilled'] = [math.nan]
        if not order.internal:
            output_df['Date Due'] = [dates[0] + pd.DateOffset(weeks=order.date_due-1)]
            for count, date in enumerate(order.date_shipped):
                output_df['Date Shipped'] = [dates[order.date_shipped[count]-1]]
                output_df['Amount Shipped'] = [order.amount_shipped[count]]
                if count < len(order.date_delivered):
                    output_df['Date Delivered'] = [dates[order.date_delivered[count]-1]]
                    output_df['Amount Delivered'] = [order.amount_delivered[count]]
                    if order.date_delivered[count] > order.date_due:
                        output_df['Late'] = [True]
                    else:
                        output_df['Late'] = [False]
                else:
                    output_df['Date Delivered'] = [math.nan]
                    output_df['Amount Delivered'] = [0]
                    output_df['Late'] = [math.nan]
                order_df = pd.concat([order_df, output_df])
        else:
            output_df['Date Due'] = [math.nan]
            output_df['Date Shipped'] = [math.nan]
```

```
                output_df['Amount Shipped'] = [math.nan]
                output_df['Date Delivered'] = [math.nan]
                output_df['Amount Delivered'] = [math.nan]
                output_df['Late'] = [False]
                order_df = pd.concat([order_df, output_df])

    return inventory_df, production_df, order_df

# %% [markdown]
# ## Initialize Period

# %%
def initPeriod(sim_period, node, dates):
    # initialize inventories and capacities for the period
    for key, product in node.finished_goods.items():
        product.inventory[sim_period] += product.inventory[sim_period-1]
        if product.annual_demand > 0:
            product.time_to_survive[sim_period] += product.inventory[sim_period-1]/(product.annual_demand/52)
    for key, product in node.raw_materials.items():
        product.inventory[sim_period] += product.inventory[sim_period-1]
        if product.annual_demand > 0:
            product.time_to_survive[sim_period] += product.inventory[sim_period-1]/(product.annual_demand/52)
    for key, process in node.processes.items():
        process.inventory[sim_period] += process.inventory[sim_period-1]
        if process.annual_demand > 0:
            process.time_to_survive[sim_period] += process.inventory[sim_period-1]/(process.annual_demand/52)
        process.capacity_available = 0
        for m_key, machine in process.machines.items():
            if (machine['start_date'] <= sim_period) and (machine['end_date'] > sim_period):
```

```
                capacity_threshold = machine['capacity_threshold']
                week = dates[sim_period-1].week
                availability = machine['availability']
                if week <= 52:
                    availability -= node.holidays[week-1]
                    availability = max(availability, 0)
                machine['percent_available'][sim_period] += capacity_threshold * availability/machine['availability']
                machine_capacity_available = machine['capacity']*availability*machine['oee']
                machine['capacity_available'][sim_period] += math.floor(machine_capacity_available*capacity_threshold)
                process.capacity_available += machine_capacity_available

# %% [markdown]
# ## Update Demand

# %%
def updateDemand(demand, supplier, raw_material_name):
    supplier.finished_goods[raw_material_name].annual_demand += demand
    for process_name in reversed(supplier.process_sequence[raw_material_name]):
        if process_name in supplier.bill_of_materials[raw_material_name]:
            quantity = sum([qty for qty in supplier.bill_of_materials[raw_material_name][process_name].values()])
            supplier.processes[process_name].annual_demand += demand*quantity
    for c_key in supplier.bill_of_materials[raw_material_name]:
        if c_key in supplier.raw_materials:
            raw_material = supplier.raw_materials[c_key]
            quantity = sum([qty for qty in supplier.bill_of_materials[raw_material_name][c_key].values()])
            component_demand = demand*quantity
```

```python
                raw_material.annual_demand += component_demand
                if len(raw_material.suppliers) > 0:
                    for s_key, supplier_tuple in raw_material.suppliers.items():
                        updateDemand(component_demand//len(raw_material.suppliers), supplier_tuple[0], c_key)

# %% [markdown]
# ## Source

# %%
def source(sim_period, customer):
    '''
    Checks the raw materials inventory at the customer and places orders according the inventory design.
    Checks the finished goods inventory at the node and places production orders according the inventory design.
    Splits the required amount among suppliers according to the contractual agreements.
    In case of disruption, the undisrupted supplier gets preference according to its available capacity.

    Parameters
    ----------
    sim_period : int
        The time at which the orders are generated.
    customer : Node (class)
        Node of the supply chain sourcing goods.
    '''

    # determine raw material orders required
    for rm_key, raw_material in customer.raw_materials.items():
        inv_target = (raw_material.design_stock[sim_period] + raw_material.replenishment_time/2) * raw_material.annual_demand//52
        inv_on_hand = raw_material.inventory[sim_period-1]
```

```
        outstanding_outflows = 0
        for order in set(customer.orders['wip']+customer.orders['shipped']):
            if rm_key in customer.bill_of_materials[order.product.name].keys():
                for p_key, quantity_required in customer.bill_of_materials[order.product.name][rm_key].items():
                    index = customer.process_sequence[order.product.name].index(p_key)
                    outstanding_outflows += order.process_amounts_required[index] * quantity_required
        outstanding_inflows = sum([order.demand - sum(order.amount_delivered)
                                  for order in customer.orders['placed']
                                  if (order.product.name == rm_key and not order.internal)])
        inv_required = inv_target - (inv_on_hand - outstanding_outflows) - outstanding_inflows
        raw_material.inventory_gap = inv_required

        if inv_required > 0:
            # define order split
            supplier_keys = [s_key for s_key in raw_material.suppliers.keys()]
            split = len(supplier_keys)
            inv_tracker = inv_required
            order_allocation = [1] * len(supplier_keys)
            order_multiples = [1] * len(supplier_keys)
            suppliers_capacity_available = []
            suppliers_moq_multiples = []

            for s_key, item in raw_material.suppliers.items():
                supplier = item[0]
                moq = item[1]
                order_lead_time = item[2]
```

```
            delivery_lead_time = supplier.finished_goods[rm_key].customers[customer.name][1]
            supplier_inventory = 0
            supplier_capacity = 1e15
            booked_cap = 0

            for count, process_name in enumerate(supplier.process_sequence[rm_key]):
                booked_cap = sum([order.process_amounts_required[count]
                                  for order in customer.orders['placed']
                                  if (order.product.name == rm_key and order.supplier == supplier)])
                supplier_capacity = max(0, min(supplier_capacity, supplier.processes[process_name].capacity_available * max(1,(order_lead_time-delivery_lead_time)) - booked_cap))
            booked_inv = sum([order.demand - sum(order.amount_shipped)
                              for order in customer.orders['placed']
                              if (order.product.name == rm_key and order.supplier == supplier)]) - booked_cap
            supplier_inventory = max(supplier_inventory, supplier.finished_goods[rm_key].inventory[sim_period] - booked_inv)
            suppliers_capacity_available += [supplier_capacity + supplier_inventory]
            suppliers_moq_multiples += [inv_required / moq]

        if (max(suppliers_moq_multiples) < 1) or (sum(suppliers_capacity_available) == 0):
            break

        for count, cap in enumerate(suppliers_capacity_available):
```

```python
            order_allocation[count] = cap/sum(suppliers_capacity_available)
            order_multiples[count] = order_allocation[count] * suppliers_moq_multiples[count]

        while sum([1 for multiple in order_multiples if multiple >= 1]) < split:
            split -= 1
            suppliers_moq_multiples = [multiple if order_allocation[count] > 0 else 1e10 for count, multiple in enumerate(suppliers_moq_multiples)]
            if min(suppliers_moq_multiples) < 1:
                min_index = suppliers_moq_multiples.index(min(suppliers_moq_multiples))
            else:
                min_index = order_multiples.index(min(order_multiples))
            suppliers_capacity_available[min_index] = 0
            for count, cap in enumerate(suppliers_capacity_available):
                order_allocation[count] = cap/sum(suppliers_capacity_available)
                order_multiples[count] = order_allocation[count] * suppliers_moq_multiples[count]

        # assign orders
        for s_key, items in raw_material.suppliers.items():
            index = supplier_keys.index(s_key)
            if order_allocation[index] > 0:
                supplier = items[0]
                order_lead_time = items[2]
                order_id = uuid.uuid4()
                demand = math.ceil(inv_required*order_allocation[index])
                inv_tracker -= demand
                order = Order(order_id, sim_period, supplier,
```

```
customer, raw_material, demand, int(sim_period+order_lead_time))
                supplier.addOrder(order, 'received')
                customer.addOrder(order, 'placed')
        if inv_tracker > 0:
            print(customer.name, sim_period, rm_key)


# determine finished goods orders required
for fg_key, finished_goods in customer.finished_goods.items():
    inv_target = finished_goods.design_stock[sim_period] * finished_goods.annual_demand//52
    inv_on_hand = finished_goods.inventory[sim_period-1]
    outstanding_outflows = sum([order.demand - sum(order.amount_shipped)
                                        for order in (customer.orders['wip'] + customer.orders['completed'])
                                        if (order.product.name == fg_key and not order.internal)])
    outstanding_inflows = sum([order.process_amounts_required[-1]
                                        for order in set(customer.orders['wip']+customer.orders['shipped']+customer.orders['delivered'])
                                        if order.product.name == fg_key and len(order.process_amounts_required) > 0])
    inv_required = inv_target - (inv_on_hand - outstanding_outflows) - outstanding_inflows
    finished_goods.inventory_gap = inv_required

    if inv_required > 0:
        order_id = uuid.uuid4()
        order = Order(order_id, sim_period, customer, customer, finished_goods, inv_required, int(sim_period+1e6), internal=True)
        customer.addOrder(order, 'received')
        customer.addOrder(order, 'placed')

```

```
    # determine wip orders required
    for p_key, process in customer.processes.items():
        inv_target = process.design_stock[sim_period] * process.annual_demand//52
        if inv_target > 0:
            inv_on_hand = process.inventory[sim_period-1]
            outstanding_outflows = 0
            outstanding_inflows = 0
            for order in process.orders:
                product = order.product
                index = customer.process_sequence[product.name].index(p_key)
                outstanding_inflows += order.process_amounts_required[index]
                if p_key not in customer.bill_of_materials[product.name]:
                    outstanding_outflows += order.demand - sum(order.amount_shipped)
                else:
                    for next_key, quantity_required in customer.bill_of_materials[product.name][p_key].items():
                        if next_key in customer.process_sequence[product.name]:
                            next_index = customer.process_sequence[product.name].index(next_key)
                            outstanding_outflows += order.process_amounts_required[next_index] * quantity_required
            inv_required = inv_target - (inv_on_hand - outstanding_outflows) - outstanding_inflows
            process.inventory_gap = inv_required

            if inv_required > 0:
                for fg_key, process_list in customer.process_sequence.items():
                    if p_key in process_list:
                        product = customer.finished_goods[fg_key]
```

```
                        break
                order_id = uuid.uuid4()
                order = Order(order_id, sim_period, customer, customer, product, inv_required, int(sim_period+1e6), internal=True, process=process)
                customer.addOrder(order, 'received')
                customer.addOrder(order, 'placed')

    return

# %% [markdown]
# ## Make

# %%
def make(sim_period, manufacturer):
    '''
    Assigns received orders to the corresponding processes at the manufacturing node.
    Produces at each process of the node according to the available inventories and process available capacity.
    Updates finished goods inventory at node according to the production during the period.

    Parameters
    ----------
    sim_period : int
            The time at which the orders are manufactured.
    supplier : Node (class)
            Node of the supply chain making goods.

    Returns
    -------
    None
    '''

    # assign received orders to processes
```

```
    outstanding_orders = [order for order in manufacturer.orders['received'] if order.date_received == sim_period]
    outstanding_orders.sort(key = lambda order: order.date_due)

    for order in outstanding_orders:
        product = order.product
        demand_to_allocate = order.demand
        index = len(manufacturer.process_sequence[product.name]) - 1

        if not order.internal:
            inventory_gap = manufacturer.finished_goods[product.name].inventory_gap
            if inventory_gap < 0:
                manufacturer.finished_goods[product.name].inventory_gap += min(demand_to_allocate, -inventory_gap)
                demand_to_allocate = max(demand_to_allocate+inventory_gap, 0)
        elif order.process == None:
            manufacturer.finished_goods[product.name].inventory_gap -= demand_to_allocate
        else:
            index = manufacturer.process_sequence[product.name].index(order.process.name)
            manufacturer.processes[order.process.name].inventory_gap -= demand_to_allocate

        if demand_to_allocate == 0:
            order.process_amounts_required += [0]*len(manufacturer.process_sequence[product.name])
            order.date_production_start = sim_period
            order.date_production_end = sim_period
            manufacturer.removeOrder(order, order.status)
            order.status = 'completed'
            manufacturer.addOrder(order, order.status)
        else:
            manufacturer.removeOrder(order, order.status)
```

```
            order.status = 'wip'
            manufacturer.addOrder(order, order.status)
            for count, process_name in reversed(list(enumerate(
manufacturer.process_sequence[product.name]))):
                if count > index:
                    order.process_amounts_required.insert(0, 0)
                elif count == index:
                    order.process_amounts_required.insert(0,
demand_to_allocate)
                elif count < index:
                    inventory_gap = min(manufacturer.processes[
process_name].inventory_gap, 0)
                    process_demand = 0
                    for next_key, quantity_required in manufacturer.
bill_of_materials[product.name][process_name].items():
                        if next_key in manufacturer.process_sequence
[product.name]:
                            next_index = manufacturer.
process_sequence[product.name].index(next_key)
                            if next_index <= index:
                                process_demand += order.
process_amounts_required[next_index-(count+1)] *
quantity_required
                    if inventory_gap < 0:
                        manufacturer.processes[process_name].
inventory_gap += min(process_demand, -inventory_gap)
                        process_demand = max(process_demand+
inventory_gap, 0)
                    order.process_amounts_required.insert(0,
process_demand)
                if order.process_amounts_required[0] > 0:
                    manufacturer.processes[process_name].orders.
append(order)

    # produce at each process
    for p_key, process in manufacturer.processes.items():
```

```
    if process.capacity_available > 0:
        for order in process.orders:
            product = order.product
            index = manufacturer.process_sequence[product.name].index(p_key)
            production_amount_required = order.process_amounts_required[index]

            if production_amount_required > 0:
                period_production = 0
                dummy_limit = 1e15
                inventory_limit = 0

                # validate how much inventory is available to produce
                for component_name in manufacturer.bill_of_materials[product.name]:
                    if p_key in manufacturer.bill_of_materials[product.name][component_name].keys():
                        if component_name in manufacturer.raw_materials.keys():
                            inventory_availabe = manufacturer.raw_materials[component_name].inventory[sim_period]
                            quantity_required = manufacturer.bill_of_materials[product.name][component_name][p_key]
                        else:
                            inventory_availabe = manufacturer.processes[component_name].inventory[sim_period]
                            quantity_required = manufacturer.bill_of_materials[product.name][component_name][p_key]
                        inventory_limit = min(dummy_limit, inventory_availabe//quantity_required)
                        dummy_limit = inventory_limit
                        if inventory_limit == 0:
                            break

```

```
            # produce according to machine capacities
            for m_key, machine in process.machines.items():
                if inventory_limit > 0 and production_amount_required > 0:
                    machine_capacity_available = machine['capacity_available'][sim_period] - machine['production'][sim_period]
                    if machine_capacity_available > 0:
                        production = min(production_amount_required, inventory_limit, machine_capacity_available)
                        machine['production'][sim_period] += production
                        machine['utilization'][sim_period] += production/machine['capacity_available'][sim_period]
                        production_amount_required -= production
                        inventory_limit -= production
                        period_production += production

        # make changes according to production
        if period_production > 0:
            order.process_amounts_required[index] -= period_production

            # reduce raw material/preceeding process inventory according to production
            for component_name in manufacturer.bill_of_materials[product.name]:
                if p_key in manufacturer.bill_of_materials[product.name][component_name].keys():
                    if component_name in manufacturer.raw_materials.keys():
                        quantity_required = manufacturer.bill_of_materials[product.name][component_name][p_key]
                        manufacturer.raw_materials[
```

```
component_name].inventory[sim_period] -= period_production*quantity_required
                        else:
                            quantity_required = manufacturer.bill_of_materials[product.name][component_name][p_key]
                            manufacturer.processes[component_name].inventory[sim_period] -= period_production*quantity_required

                # update finished goods inventory
                if index + 1 == len(manufacturer.process_sequence[product.name]):
                    manufacturer.finished_goods[product.name].inventory[sim_period] += period_production
                    order.amount_produced += period_production
                else:
                    process.inventory[sim_period] += period_production

                # change order status
                if order.date_production_start == 0:
                    order.date_production_start = sim_period

                if production_amount_required == 0:
                    process.orders.remove(order)
                    if sum(order.process_amounts_required) == 0:
                        if order.internal:
                            order.date_fulfilled = sim_period
                        order.date_production_end = sim_period
                        if order.status != ('shipped' or 'delivered'):
                            manufacturer.removeOrder(order, order.status)
                            order.status = 'completed'
                            manufacturer.addOrder(order, order.status)
```

```
    return

# %% [markdown]
# ## Deliver

# %%
def deliver(sim_period, supplier):
    '''
    Ships and delivers finished goods according to the supplier's available inventory and delivery lead times.
    Registers shipped and delivered orders accordingly.
    Updates finished goods inventory of the supplier according to shipments.
    Updates raw materials inventory of customer according to deliveries.

    Parameters
    ----------
    sim_period : int
            The time at which the orders are shipped/delivered.
    supplier : Node (class)
            Node of the supply chain delivering goods.

    Returns
    -------
    None
    '''

    # check orders that need to be shipped at the beginning of the period
    orders_to_be_shipped = []
    for order in supplier.orders['completed'] + supplier.orders['wip'] + supplier.orders['received']:
        if order.customer.name != supplier.name:
            delivery_lead_time = supplier.finished_goods[order.product.name].customers[order.customer.name][1]
```

```
            if sim_period >= order.date_due - delivery_lead_time - 1:
                orders_to_be_shipped.append(order)

    # update supplier's inventories and order shipped amount
    for order in orders_to_be_shipped:
        ship_amount = min(supplier.finished_goods[order.product.name].inventory[sim_period], order.demand - sum(order.amount_shipped))
        if ship_amount > 0:
            supplier.finished_goods[order.product.name].inventory[sim_period] -= ship_amount
            order.amount_shipped += [ship_amount]
            order.date_shipped += [sim_period]

            if order not in supplier.orders['shipped']:
                supplier.addOrder(order, 'shipped')

            # when shipped in full, update order status
            if order.demand == sum(order.amount_shipped):
                supplier.removeOrder(order, order.status)
                order.status = 'shipped'


    # check orders in transit and update customer's inventories after delivery
    orders_to_be_delivered = []
    for order in supplier.orders['shipped']:
        delivery_lead_time = supplier.finished_goods[order.product.name].customers[order.customer.name][1]
        for date_shipped in order.date_shipped:
            if sim_period - delivery_lead_time == date_shipped:
                index = order.date_shipped.index(sim_period - delivery_lead_time)
                if sum(order.amount_delivered) == 0:
                    order.product.late_delivery += order.demand -
```

```
order.amount_shipped[index]
                order.amount_delivered += [order.amount_shipped[index]]
                order.date_delivered += [sim_period]
                order.product.inventory[sim_period] += order.amount_shipped[index]
                if sum(order.amount_delivered) == order.demand:
                    orders_to_be_delivered.append(order)

    # when delivered in full, update order status
    for order in orders_to_be_delivered:
        supplier.removeOrder(order, order.status)
        order.status = 'delivered'
        supplier.addOrder(order, order.status)
        order.date_fulfilled = sim_period
    return

# %% [markdown]
# ## Disrupt

# %%
def disrupt(sim_period, disruptions, nodes_dict, consumers, demand_distortion):
    '''
    Modify capacity, inventory, design stock level, machine availability and oee, or demand at the corresponding node according to the disruption.

    Parameters
    ----------
    sim_period : int
            The time at which the disruptions starts/stops.
    disruptions : list
            List with the disruptions that in the corresponding simulated scenario.
    nodes_dict : dict
```

```
            Dictionary with all Nodes (class) involved in the simulation.
    consumers : list
            List with the names of the end consumer Nodes.
    demand_distortion : list
            List with the demand distortions that applies for each end consumer.

    Returns
    -------
    None
    '''

    for disruption in disruptions:
        if disruption.date_start == sim_period:
            node = nodes_dict[disruption.node_name]
            if disruption.type == 'capacity':
                if disruption.process_name == '':
                    for p_key, process in node.processes.items():
                        for m_key, machine in process.machines.items():
                            machine['capacity_threshold'] = disruption.capacity_threshold
                else:
                    process = node.processes[disruption.process_name]
                    if disruption.machine_name == '':
                        for m_key, machine in process.machines.items():
                            machine['capacity_threshold'] = disruption.capacity_threshold
                    else:
                        process.machines[disruption.machine_name]['capacity_threshold'] = disruption.capacity_threshold

```

```
                elif disruption.type == 'demand':
                    consumer_names = [consumer.name for consumer in consumers]
                    index = consumer_names.index(disruption.node_name)
                    demand_distortion[index] += disruption.demand_variation

                elif disruption.type == 'policy':
                    if disruption.product_name != '':
                        if disruption.product_name in node.finished_goods.keys():
                            product = node.finished_goods[disruption.product_name]
                            product.design_stock[sim_period:] = [disruption.design_stock]*len(product.design_stock[sim_period:])
                        elif disruption.product_name in node.raw_materials.keys():
                            product = node.raw_materials[disruption.product_name]
                            product.design_stock[sim_period:] = [disruption.design_stock]*len(product.design_stock[sim_period:])
                    elif disruption.process_name != '':
                        process = node.processes[disruption.process_name]
                        process.design_stock[sim_period:] = [disruption.design_stock]*len(process.design_stock[sim_period:])

                elif disruption.type == 'inventory':
                    if disruption.product_name == '' and disruption.process_name == '':
                        for fg_key, product in node.finished_goods.items():
                            product.inventory[sim_period] -= math.ceil(product.inventory[sim_period-1]*disruption.inventory_loss)
                        for rm_key, product in node.raw_materials.
```

```
items():
                        product.inventory[sim_period] -= math.ceil(product.inventory[sim_period-1]*disruption.inventory_loss)
                    for p_key, process in node.processes.items():
                        process.inventory[sim_period] -= math.ceil(process.inventory[sim_period-1]*disruption.inventory_loss)
                elif disruption.product_name != '':
                    if disruption.product_name in node.finished_goods.keys():
                        product = node.finished_goods[disruption.product_name]
                        product.inventory[sim_period] -= math.ceil(product.inventory[sim_period-1]*disruption.inventory_loss)
                    elif disruption.product_name in node.raw_materials.keys():
                        product = node.raw_materials[disruption.product_name]
                        product.inventory[sim_period] -= math.ceil(product.inventory[sim_period-1]*disruption.inventory_loss)
                elif disruption.process_name != '':
                    process = node.processes[disruption.process_name]
                    process.inventory[sim_period] -= math.ceil(process.inventory[sim_period-1]*disruption.inventory_loss)

            elif disruption.type == 'machine':
                process = node.processes[disruption.process_name]
                process.machines[disruption.machine_name]['availability'] = disruption.machine_availability
                process.machines[disruption.machine_name]['oee'] = disruption.machine_oee


        elif disruption.date_end == sim_period:
```

```
                node = nodes_dict[disruption.node_name]
                if disruption.type == 'capacity':
                    if disruption.process_name == '':
                        for p_key, process in node.processes.items():
                            for m_key, machine in process.machines.items():
                                machine['capacity_threshold'] = 1
                    else:
                        process = node.processes[disruption.process_name]
                        if disruption.machine_name == '':
                            for m_key, machine in process.machines.items():
                                machine['capacity_threshold'] = 1
                        else:
                            process.machines[disruption.machine_name]['capacity_threshold'] = 1

                elif disruption.type == 'demand':
                    consumer_names = [consumer.name for consumer in consumers]
                    index = consumer_names.index(disruption.node_name)
                    demand_distortion[index] -= disruption.demand_variation
    return demand_distortion

# %% [markdown]
# # Inputs (Read Excel)

# %%
df = pd.read_excel('Input Template.xlsm', sheet_name=None)

# %%
df['Holidays'] = df['Holidays'].fillna(0)
```

```
# %% [markdown]
# ## Variables

# %%
nodes_dict = {}
consumers = []
leaf_nodes = []
max_lead_time = 0
demand_distortion = []
machine_updates = []
sim_start = df['Demand']['Date'].iloc[0]

# %% [markdown]
# ### Supply Chain Nodes

# %% [markdown]
# #### All Nodes

# %%
for index, row in df['Node Keys'].iterrows():
    node_name = row.iloc[0]
    nodes_dict[node_name] = Node(node_name)
    nodes_dict[node_name].holidays += df['Holidays'][node_name].to_list()

# %% [markdown]
# #### End Consumer Nodes

# %%
for count, column_name in enumerate(df['Demand'].columns):
    if count > 1:
        consumers += [nodes_dict[column_name]]
        demand_distortion += [1]

# %% [markdown]
```

```
# #### Leaf Nodes

# %%
for n_key, node in nodes_dict.items():
    if df['Raw Materials'].loc[df['Raw Materials'].iloc[:,0].isin([
   n_key])].size == 0:
        leaf_nodes += [node]

# %% [markdown]
# ### Processes

# %%
for index, row in df['Processes'].iterrows():
    node_name = row.iloc[0]
    process_name = row.iloc[1]
    generic = row.iloc[2]
    initial_inventory = row.iloc[3]
    design_stock = row.iloc[4]

    process = Process(process_name, initial_inventory, design_stock,
    generic)
    nodes_dict[node_name].addProcess(process)

# %% [markdown]
# #### Machines

# %%
for index, row in df['Machines'].iterrows():
    node_name = row.iloc[0]
    process_name = row.iloc[1]
    machine_name = row.iloc[2]
    capacity = row.iloc[3]
    availability = row.iloc[4]
    oee = row.iloc[5]
    start_date = row.iloc[6]
    end_date = row.iloc[7]
```

```
    start_date = max(0, (start_date.to_pydatetime() - sim_start.
   to_pydatetime()).days//7 + 1)
    end_date = max(0, (end_date.to_pydatetime() - sim_start.
   to_pydatetime()).days//7 + 1)

    process = nodes_dict[node_name].processes[process_name]
    process.addMachine(machine_name, capacity, availability, oee,
   start_date, end_date)

# %% [markdown]
# #### Machine Updates

# %%
for index, row in df['Machine Updates'].iterrows():
    node_name = row.iloc[0]
    if not isinstance(node_name,str):
        continue
    process_name = row.iloc[1]
    machine_name = row.iloc[2]
    availability = row.iloc[3]
    oee = row.iloc[4]
    change_date = row.iloc[5]

    change_date = max(0, (change_date.to_pydatetime() - sim_start.
   to_pydatetime()).days//7 + 1)
    machine_updates += [(node_name, process_name, machine_name,
   availability, oee, change_date)]

# %% [markdown]
# ### Products

# %% [markdown]
# #### Raw Materials

# %%
```

```python
for index, row in df['Raw Materials'].iterrows():
    node_name = row.iloc[0]
    product_name = row.iloc[1]
    initial_inventory = row.iloc[2]
    design_stock = row.iloc[3]

    product = Product(product_name,initial_inventory,design_stock)
    nodes_dict[node_name].addRawMaterial(product)

# %% [markdown]
# #### Finished Goods

# %%
for index, row in df['Finished Goods'].iterrows():
    node_name = row.iloc[0]
    product_name = row.iloc[1]
    initial_inventory = row.iloc[2]
    design_stock = row.iloc[3]
    product = Product(product_name,initial_inventory,design_stock)
    nodes_dict[node_name].addFinishedGoods(product)

# %% [markdown]
# #### Suppliers and Customers

# %%
for index, row in df['Suppliers'].iterrows():
    customer_name = row.iloc[0]
    product_name = row.iloc[1]
    supplier_name = row.iloc[2]
    minimum_order_quantity = row.iloc[3]
    order_lead_time = row.iloc[4]
    delivery_lead_time = row.iloc[5]
    accelerated_lead_time = row.iloc[6]

    max_lead_time = max(order_lead_time, max_lead_time)

```

```
    customer = nodes_dict[customer_name]
    supplier = nodes_dict[supplier_name]

    raw_material = customer.raw_materials[product_name]
    raw_material.addSupplier(supplier, minimum_order_quantity,
   order_lead_time, accelerated_lead_time)

    finished_good = supplier.finished_goods[product_name]
    finished_good.addCustomer(customer, delivery_lead_time)

# %% [markdown]
# ### Bill of Materials

# %%
for index, row in df['BOM'].iterrows():
    node_name = row.iloc[0]
    product_name = row.iloc[1]
    raw_material_name = row.iloc[2]
    process_name = row.iloc[3]
    quantity = row.iloc[4]
    step_name = row.iloc[5]

    if isinstance(raw_material_name, str):
        component_name = raw_material_name
    else:
        component_name = process_name

    nodes_dict[node_name].addBOM(product_name, component_name,
   quantity, step_name)

# %% [markdown]
# ### Process Sequence

# %%
for node_name, node in nodes_dict.items():
    for finished_good_name in node.finished_goods.keys():
```

```
        process_sequence = []
        filtered_list = df['BOM'].loc[df['BOM'].iloc[:,0].isin([node_name])]
        filtered_list = filtered_list.loc[df['BOM'].iloc[:,1].isin([finished_good_name])]
        process_set = set([row.iloc[5] for index, row in filtered_list.iterrows()])

        if len(process_set) == 1:
            node.addProcessSequence(finished_good_name, [*process_set, ])
        elif len(process_set) > 1:
            while len(process_set) > 0:
                for process_name in process_set:
                    short_list = filtered_list.loc[df['BOM'].iloc[:,3].isin([process_name])]
                    short_list = set([row.iloc[3] for index, row in short_list.iterrows()])
                    if len(short_list) == 0:
                        process_set.discard(process_name)
                        process_sequence.insert(0,process_name)
                        filtered_list = filtered_list.loc[df['BOM'].iloc[:,5].isin([*process_set,])]
                        break
            node.addProcessSequence(finished_good_name, process_sequence)

# %% [markdown]
# ## Parameters

# %%
sim_duration = 0
sim_years = []
weeks = []
dates = []

```

```
# %%
for index, row in df['Demand'].iterrows():
    if not row.iloc[0]:
        break
    date = row.iloc[1]
    weeks += [(date.to_pydatetime() - sim_start.to_pydatetime()).
    days//7 + 1]
weeks += [weeks[-1]+4]

# %%
sim_duration = max_lead_time + weeks[-1]

# %%
dates = [sim_start]
for period in range(1,sim_duration):
    dates += [dates[0] + pd.DateOffset(weeks=period)]

# %% [markdown]
# ## Disruptions

# %%
disruption_scenarios = {'Baseline':[]}

# %%
for index, row in df['Disruption Parameters'].iterrows():
    include = row.iloc[0]
    scenario_name = row.iloc[1]
    if not include or not isinstance(scenario_name,str):
        continue
    date_start = (row.iloc[2].to_pydatetime() - sim_start.
    to_pydatetime()).days//7 + 1
    date_end = (row.iloc[3].to_pydatetime() - sim_start.
    to_pydatetime()).days//7 + 1
    node_name = row.iloc[4]
    product_name = row.iloc[5]
    process_name = row.iloc[6]
```

```
        machine_name = row.iloc[7]
        capacity_threshold = row.iloc[8]
        inventory_loss = row.iloc[9]
        demand_variation = row.iloc[10]
        design_stock = row.iloc[11]
        new_availability = row.iloc[12]
        new_oee = row.iloc[13]

        if not isinstance(product_name,str):
            product_name = ''
        if not isinstance(process_name,str):
            process_name = ''
        if not isinstance(machine_name,str):
            machine_name = ''

        if not math.isnan(capacity_threshold):
            disruption_type = 'capacity'
            disruption = Disruption(date_start, date_end, node_name, product_name, process_name,
                                    machine_name, capacity_threshold=capacity_threshold, type=disruption_type)
        elif not math.isnan(inventory_loss):
            disruption_type = 'inventory'
            disruption = Disruption(date_start, date_start, node_name, product_name, process_name,
                                    inventory_loss=inventory_loss, type=disruption_type)
        elif not math.isnan(demand_variation):
            disruption_type = 'demand'
            disruption = Disruption(date_start, date_end, node_name, product_name, process_name,
                                    machine_name, demand_variation=demand_variation, type=disruption_type)
        elif not math.isnan(design_stock):
            disruption_type = 'policy'
            disruption = Disruption(date_start, date_start, node_name,
```

```
product_name, process_name,
                                design_stock=design_stock, type=disruption_type)
    elif not math.isnan(new_availability) and not math.isnan(new_oee):
        disruption_type = 'machine'
        disruption = Disruption(date_start, date_start, node_name, product_name, process_name, machine_name,
                                new_availability=new_availability, new_oee=new_oee, type=disruption_type)



    if scenario_name not in disruption_scenarios.keys():
        disruption_scenarios[scenario_name] = [disruption]
    else:
        disruption_scenarios[scenario_name] += [disruption]

# %% [markdown]
# # Simulation

# %% [markdown]
# ### Initialize Simulation (only once)

# %%
initSim(sim_duration, nodes_dict)
inventory_df = pd.DataFrame()
production_df = pd.DataFrame()
order_df = pd.DataFrame()

# %% [markdown]
# ### Reset Simulation (only if running the logic block for a second + time without restarting kernel)

# %%
# demand_distortion = resetSim(sim_duration, nodes_dict,
```

```
    demand_distortion)
# inventory_df = pd.DataFrame()
# production_df = pd.DataFrame()
# order_df = pd.DataFrame()

# %% [markdown]
# ### Logic

# %%
demand_penalty = 0 # decimal

print(f"Simulation started at {datetime.datetime.now()}.")
for simulation_scenario, disruptions in disruption_scenarios.items()
    :
    for sim_period in range(1,sim_duration+1):

        # UPDATE ANNUAL DEMAND
        if sim_period in weeks[:-13]:
            index = weeks.index(sim_period)
            for node in nodes_dict.values():
                for p_key, process in node.processes.items():
                    process.annual_demand = 0
                for key, raw_material in node.raw_materials.items():
                    raw_material.annual_demand = 0
                for key, finished_good in node.finished_goods.items
    ():
                    finished_good.annual_demand = 0
            for consumer in consumers:
                for rm_key, raw_material in consumer.raw_materials.
    items():
                    demand = max(sum(df['Demand'][consumer.name].
    iloc[index+1:index+13]) - math.floor(raw_material.late_delivery*
    demand_penalty), 0)
                    raw_material.annual_demand += demand
                    if len(raw_material.suppliers) > 0:
                        for s_key, supplier in raw_material.
```

```
suppliers.items():
                        updateDemand(demand//len(raw_material.suppliers), supplier[0], rm_key)

    # UPDATE MACHINES
    for item in filter(lambda x: x[-1] == sim_period, machine_updates):
        nodes_dict[item[0]].processes[item[1]].machines[item[2]]['availability'] = item[3]
        nodes_dict[item[0]].processes[item[1]].machines[item[2]]['oee'] = item[4]

    # APPLY DISRUPTIONS
    demand_distortion = disrupt(sim_period, disruptions, nodes_dict, consumers, demand_distortion)

    # GENERATE DEMAND AND DISCOUNT PENALTIES (undelivered)
    if sim_period in weeks[:-1]:
        index = weeks.index(sim_period)
        for count, consumer in enumerate(consumers):
            for rm_key, raw_material in consumer.raw_materials.items():
                for s_key, items in raw_material.suppliers.items():
                    supplier = items[0]
                    order_lead_time = items[2]
                    demand = max(df['Demand'][consumer.name][index] * demand_distortion[count] - math.floor(raw_material.late_delivery*demand_penalty), 0)
                    order_id = uuid.uuid4()
                    order = Order(order_id, sim_period, supplier, consumer, raw_material, demand, sim_period+order_lead_time)
                    supplier.addOrder(order, 'received')
                    customer.addOrder(order, 'placed')

    # INITIALIZE INVENTORY AND CAPACITY FOR PERIOD
```

```
        for node in nodes_dict.values():
            initPeriod(sim_period, node, dates)

        # SOURCE MATERIALS
        for node in nodes_dict.values():
            if node not in consumers+leaf_nodes:
                source(sim_period, node)

        # MAKE PRODUCTS
        for node in nodes_dict.values():
            make(sim_period, node)

        # DELIVER GOODS
        for node in nodes_dict.values():
            deliver(sim_period, node)

    inventory_df, production_df, order_df = recordSim(simulation_scenario, sim_duration, nodes_dict,
                                                       dates, inventory_df, production_df, order_df)
    demand_distortion = resetSim(sim_duration, nodes_dict, demand_distortion)
    print(f"Finished running the '{simulation_scenario}' scenario at {datetime.datetime.now()}")
    print(f"... {len(disruption_scenarios) - list(disruption_scenarios.keys()).index(simulation_scenario) - 1} scenarios remaining")

# %% [markdown]
# # Outputs (Write Excel)

# %%
print(f"Started printing outputs at {datetime.datetime.now()}")
with pd.ExcelWriter("Output Template.xlsx",
                    mode="a",
                    engine="openpyxl",
```

```
                    if_sheet_exists="replace",
                    ) as writer:
        inventory_df.to_excel(writer, sheet_name="Inventory")
        production_df.to_excel(writer, sheet_name="Production")
        order_df.to_excel(writer, sheet_name="Orders")
print(f"Finished printing outputs at {datetime.datetime.now()}")

# %%
```

THIS PAGE INTENTIONALLY LEFT BLANK

Appendix C

Front End Appendix

C.1 Input File Tabs

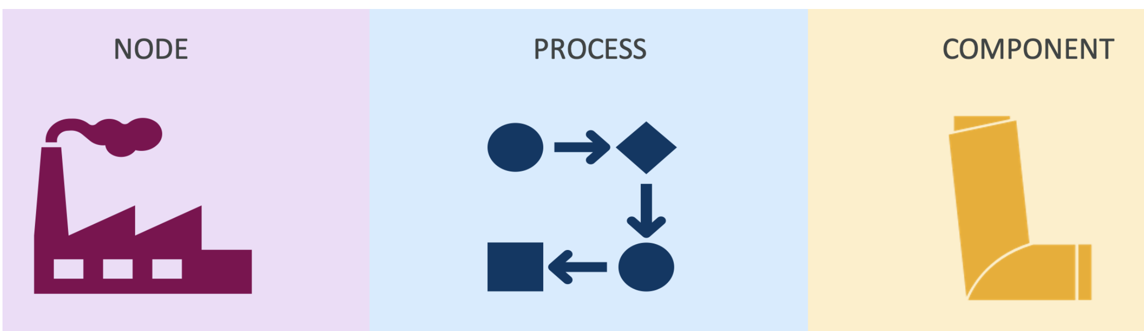


Figure C-1: Reconfigure Supply Chain Tab

Run?	Scenario	Start Date	End Date	Node	Component	Process	Machine	Capacity Threshold (%)	Inventory Loss (%)	Demand Variation (%)	New Design Stock (WOS)	New Availability (days/week)	New OEE (%)
FALSE	Example												
FALSE		Jan-25	Jan-26		[redacted]			0%	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
FALSE		Jan-24	[redacted]			[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	80	[redacted]	[redacted]
FALSE		Jan-26	[redacted]		[redacted]			[redacted]	[redacted]	[redacted]	[redacted]	7	85%
FALSE		Jan-26	Jul-26		[redacted]			0%	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
FALSE		Jan-26	Jul-26		[redacted]			0%	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
FALSE		Jan-25	[redacted]			[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	32	[redacted]	[redacted]
FALSE		Jan-26	[redacted]			[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	22	[redacted]	[redacted]
FALSE		Jan-26	Jul-26		[redacted]			0%	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
FALSE		Jan-26	Jul-26		[redacted]			0%	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
FALSE		Jan-26	Jul-26		[redacted]			0%	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
FALSE		Jan-26	Jul-26		[redacted]			0%	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]

Figure C-2: Disruption Parameters Tab

NODE	PROCESS	INITIAL INVENTORY (units of inventory sitting at the end of the process)	DESIGN STOCK (weeks of supply)
		0	0
		7,401,839	8
		0	0
		0	0
		6,436,799	2
		5,241,461	2
		8,252,313	2
		6,431,465	2
		8,523,494	2
		0	0
		6,370,736	2
		9,058,502	2
		7,030,483	2
		0	0
		9,780,679	2
		8,368,477	2
		7,594,800	2
		9,662,019	2
		8,514,937	2
		0	0
		8,685,468	2
		9,915,226	2
		6,883,651	2
		0	0
		0	0
		0	0

Figure C-3: Processes Tab

NODE	PROCESS	MACHINE NAME	INSTALLED CAPACITY (units/day)	AVAILABILITY (days/week)	OEE (%)	START DATE	END DATE
			398,147	7	34%	1-Jan-26	31-Dec-35
			295,651	6	44%	1-Jan-30	31-Dec-35
			131,341	7	90%	1-Jan-24	31-Dec-35
			367,435	6	40%	1-Jan-28	31-Dec-35
			260,961	5	44%	1-Jan-24	31-Dec-35
			112,004	5	64%	1-Jan-24	31-Dec-35
			304,711	5	76%	1-Jan-24	31-Dec-35
			181,304	5	79%	1-Apr-24	31-Dec-35
			298,865	5	76%	1-Jul-26	31-Dec-35
			171,604	5	75%	1-Oct-26	31-Dec-35
			406,559	5	45%	1-Oct-28	31-Dec-35
			446,235	6	35%	1-Oct-28	31-Dec-35
			406,624	6	62%	1-Oct-29	31-Dec-35
			305,247	7	71%	1-Jan-24	31-Dec-35
			498,974	5	30%	1-Jan-25	31-Dec-35
			187,494	6	41%	1-Jan-28	31-Dec-35
			391,907	7	55%	1-Jan-24	31-Dec-35
			126,257	5	53%	1-Jan-25	31-Dec-35
			187,393	6	89%	1-Jan-28	31-Dec-35
			211,485	6	66%	1-Jan-24	31-Dec-35
			111,701	7	59%	1-Jan-25	31-Dec-35
			121,030	6	35%	1-Jan-28	31-Dec-35
			340,283	5	55%	1-Jan-24	31-Dec-35
			328,873	7	87%	1-Jan-25	31-Dec-35
			210,298	7	47%	1-Jan-28	31-Dec-35
			113,755	6	90%	1-Jan-24	31-Dec-35

Figure C-4: Machines Tab

NODE	PROCESS	MACHINE NAME	NEW AVAILABILITY (days/week)	NEW OEE (%)	DATE OF CHANGE
			5	39%	1-Jan-25
			5	39%	1-Jan-26
			5	52%	1-Jan-27
			5	53%	1-Jan-28
			5	54%	1-Jan-29
			5	55%	1-Jan-30
			5	56%	1-Jan-31
			5	57%	1-Jan-32
			5	61%	1-Jan-28
			7	62%	1-Jan-29
			5	63%	1-Jan-30
			5	64%	1-Jan-31
			5	65%	1-Jan-32
			5	47%	1-Jan-25
			5	52%	1-Jan-26
			5	55%	1-Jan-27
			5	56%	1-Jan-28
			5	57%	1-Jan-29
			5	58%	1-Jan-30
			5	59%	1-Jan-31
			5	60%	1-Jan-32
			5	73%	1-Jan-27
			5	90%	1-Jan-28
			5	75%	1-Jan-31
			5	90%	1-Jan-32

Figure C-5: Machine Updates Tab

NODE	COMPONENT	INITIAL INVENTORY (units of inventory)	DESIGN STOCK (weeks of supply)
		6,348,598	7
		9,162,479	7
		18,471,427	7
		13,299,990	7
		2,831,546	7
		19,182,125	8
		10,592,584	10
		5,004,664	36
		15,762,257	21
		14,639,816	22
		11,370,576	42
		17,645,636	30
		5,224,657	30
		16,105,100	30
		11,562,160	39
		6,835,870	4
		1,022,891	4
		11,676,486	30
		11,810,885	10
		11,468,114	78
		1,085,688	26
		11,013,265	26
		14,096,270	26
		1,935,942	26
		3,265,690	26

Figure C-6: Raw Materials Tab

NODE	COMPONENT	INITIAL INVENTORY (units of inventory)	DESIGN STOCK (weeks of supply)
		201,516	9
		3,555,086	9
		0	0
		1,000,000,000,000	52
		1,000,000,000,000	52
		1,000,000,000,000	52
		472	19
		472	19
		3,711,131	18
		4,948,175	24
		4,123,479	10
		4,948,175	12
		3,298,783	8
		1,855,565	9
		1,855,565	9
		1,855,565	9
		1,855,565	9
		4,948,175	12
		3,448,175	12
		0	0
		1,649,391	4
		1,000,000,000,000	0
		1,000,000,000,000	0
		1,000,000,000,000	0
		1,000,000,000,000	0

Figure C-7: Finished Goods Tab

CUSTOMER (node)	COMPONENT	SUPPLIER (node)	MINIMUM ORDER QUANTITY (units)	ORDER LEAD TIME (weeks)	DELIVERY LEAD TIME (weeks)
			13,000	20	2
			55,000	16	2
			36,000	5	3
			32,800	13	3
			46,200	15	3
			75,800	20	1
			33,400	14	1
			56,800	7	1
			79,600	8	2
			16,400	12	2
			68,200	13	3
			23,600	5	2
			91,400	15	3
			18,600	19	3
			98,200	8	2
			13,600	9	2
			48,200	18	2
			19,400	17	2
			89,000	11	1
			29,600	11	3
			59,000	13	2
			3,800	13	3
			22,600	9	2
			34,000	14	2
			12,800	19	3

Figure C-8: Suppliers Tab

NODE	FINISHED GOOD	REQUIRED RAW MATERIAL	REQUIRED PREVIOUS PROCESS	QUANTITY (BUOM)	PROCESS TO PERFORM
	Breztri			1	Packing
	Breztri			5.48013E-05	Filling
	Breztri			1	Filling
	Breztri			1	Filling
	Breztri			1	Filling
	Breztri			1	Filling
	Breztri			1	Filling
	Breztri			1	Filling
	Breztri			1	Assembly
	Breztri			1	Assembly
	Breztri			1	Assembly
	Breztri			1	Assembly
	Breztri			1	Assembly
	Breztri		Filling	1	Assembly
	Breztri			1	Packing
	Breztri			1	Packing
	Breztri		Assembly	1	Packing
	Filled Cans Breztri			5.48013E-05	Filling
	Filled Cans Breztri			1	Filling
	Filled Cans Breztri			1	Filling
	Filled Cans Breztri			1	Filling
	Filled Cans Breztri			1	Filling
	Filled Cans Breztri			1	Filling
	Filled Cans Breztri			1	Filling
	Filled Cans Breztri			1	Assembly
	Filled Cans Breztri			1	Assembly
	Filled Cans Breztri			1	Assembly
	Filled Cans Breztri			1	Assembly
	Filled Cans Breztri			1	Assembly
	Filled Cans Breztri		Filling	1	Assembly
	Canister uncoated			1	Deep Drawing
	Canister uncoated		Deep Drawing	1	Washing
	Canister			1	Coating
	Canister			1	Coating

Figure C-9: Bill of materials Tab

Run?	Date	Global Breztri consumer	Japan Breztri consumer	China Breztri consumer	EU Breztri consumer	USA Breztri consumer	Global Bevespi consumer	Global Airsupra consumer
TRUE	Jan-24							
TRUE	Feb-24							
TRUE	Mar-24							
TRUE	Apr-24							
TRUE	May-24							
TRUE	Jun-24							
TRUE	Jul-24							
TRUE	Aug-24							
TRUE	Sep-24							
TRUE	Oct-24							
TRUE	Nov-24							
TRUE	Dec-24							
TRUE	Jan-25							
TRUE	Feb-25							
TRUE	Mar-25							
TRUE	Apr-25							
TRUE	May-25							
TRUE	Jun-25							
TRUE	Jul-25							
TRUE	Aug-25							
TRUE	Sep-25							
TRUE	Oct-25							
TRUE	Nov-25							
TRUE	Dec-25							
TRUE	Jan-26							
TRUE	Feb-26							
TRUE	Mar-26							
TRUE	Apr-26							
TRUE	May-26							

Figure C-10: Demand Tab

Week	Aprox. Week Start							
23	4-Jun							
24	11-Jun							
25	18-Jun	1	1					1
26	25-Jun							
27	2-Jul			1				1
28	9-Jul				1	1		
29	16-Jul					7		
30	23-Jul					7		
31	30-Jul					7		7
32	6-Aug	1						7
33	13-Aug				1	1		
34	20-Aug						1	
35	27-Aug							
36	3-Sep							1
37	10-Sep			1				
38	17-Sep							
39	24-Sep							
40	1-Oct		1					
41	8-Oct							1
42	15-Oct							
43	22-Oct							
44	29-Oct	1			1	1		
45	5-Nov				1	1		1
46	12-Nov							
47	19-Nov							
48	26-Nov			1				7
49	3-Dec							
50	10-Dec							
51	17-Dec					7	2	
52	24-Dec	6	7	6	1	7	7	7
Total	**Non labor days**	**10**	**10**	**9**	**11**	**45**	**16**	**38**
Total	**Open days**	**354**	**354**	**355**	**353**	**319**	**348**	**326**
Total	**Open weeks**	**50.6**	**50.6**	**50.7**	**50.4**	**45.6**	**49.7**	**46.6**

Figure C-11: Holidays Tab

C.2 Input File Code

Input Template.xlsm - AddNodeForm (Code)

CommandButton1 | Click

```vba
Private Sub CommandButton1_Click()
    Dim ws As Worksheet
    Dim targetSheet As Worksheet
    Dim lastRow As Long
    Dim textBoxValue As String
    Dim cell As Range
    Dim foundCell As Range
    Dim nodeIndex As Long

    textBoxValue = Me.TextBox1.Value

    Set ws = ThisWorkbook.Sheets("Node Keys")
    lastRow = ws.Cells(ws.Rows.Count, "A").End(xlUp).Row

    'Check if the new Node has a name
    If textBoxValue = "" Then
        MsgBox "Please name the Node you want to add first"
        Exit Sub
    End If

    'Check if Node already exists
    Set searchRange = ws.Range("A2:A" & lastRow)
    For Each cell In searchRange
        If cell.Value = textBoxValue Then
            MsgBox "That Node already exists in the model"
            Exit Sub
        End If
    Next cell

    'Add Node and sort alphabetcally
    ws.Cells(lastRow + 1, "A").Value = textBoxValue

    ws.Sort.SortFields.Clear
    ws.Sort.SortFields.Add Key:=Range("A1"), _
        SortOn:=xlSortOnValues, Order:=xlAscending, DataOption:=xlSortNormal
    With ws.Sort
        .SetRange ws.Range("A1:A" & lastRow + 1)
        .Header = xlYes
        .MatchCase = False
        .Apply
    End With
```

Input Template.xlsm - AddNodeForm (Code)

CommandButton1 | Click

```vba
        .SetRange ws.Range("A1:A" & lastRow + 1)
        .Header = xlYes
        .MatchCase = False
        .Apply
    End With

    Set searchRange = ws.Range("A2:A" & lastRow + 1)
    Set foundCell = searchRange.Find(textBoxValue, _
        LookIn:=xlValues, lookat:=xlWhole)

    nodeIndex = foundCell.Row - 1

    'Add Node to other tabs
    Set targetSheet = ThisWorkbook.Sheets("Process Keys")
    targetSheet.Columns(nodeIndex).Insert
    targetSheet.Range(targetSheet.Cells(1, nodeIndex - 1).Address()).Copy
    targetSheet.Range(targetSheet.Cells(1, nodeIndex).Address()).PasteSpecial xlPasteAll

    Set targetSheet = ThisWorkbook.Sheets("Component Keys")
    targetSheet.Columns(nodeIndex).Insert
    targetSheet.Range(targetSheet.Cells(1, nodeIndex - 1).Address()).Copy
    targetSheet.Range(targetSheet.Cells(1, nodeIndex).Address()).PasteSpecial xlPasteAll

    Set targetSheet = ThisWorkbook.Sheets("Holidays")
    targetSheet.Columns(nodeIndex + 2).Insert
    targetSheet.Range(targetSheet.Cells(1, nodeIndex + 1).Address()).Copy
    targetSheet.Range(targetSheet.Cells(1, nodeIndex + 2).Address()).PasteSpecial xlPasteAll

    ' Update Name dropdown
    With ActiveWorkbook.Names("NODES")
        .Name = "NODES"
        '.RefersToR1C1 = "='Node Keys'!R2C1:R1048576C1"
        .RefersToR1C1 = ws.Range("A2", "A" & lastRow + 1)
        .Comment = ""
    End With

    MsgBox "Node added successfully"

    Unload Me

End Sub
```

Figure C-12: VBA Form for adding nodes

Input Template.xlsm - AddComponentForm (Code)

CommandButton1 | Click

```vba
    Set foundCell = searchRange.Find(comboBoxValue, _
        LookIn:=xlValues, lookat:=xlWhole)

    nodeIndex = foundCell.Row - 1

    Set ws = ThisWorkbook.Sheets("Component Keys")
    lastRow = ws.Cells(ws.Rows.Count, nodeIndex).End(xlUp).Row

    'Check if Component already exists
    Set searchRange = ws.Range(ws.Cells(1, nodeIndex).Address(), ws.Cells(lastRow, nodeIndex).Address())
    For Each cell In searchRange
        If cell.Value = textBoxValue Then
            MsgBox "That Component already exists within the Node"
            Exit Sub
        End If
    Next cell

    'Add Component under Node and sort alphabetically
    ws.Cells(lastRow + 1, nodeIndex).Value = textBoxValue

    Set searchRange = ws.Range(ws.Cells(2, nodeIndex).Address(), ws.Cells(lastRow + 1, nodeIndex).Address())
    ws.Sort.SortFields.Clear
    ws.Sort.SortFields.Add2 Key:=searchRange, _
        SortOn:=xlSortOnValues, Order:=xlAscending, DataOption:= _
        xlSortNormal
    With ws.Sort
        .SetRange searchRange
        .Header = xlNo
        .MatchCase = False
        .Orientation = xlTopToBottom
        .SortMethod = xlPinYin
        .Apply
    End With

    MsgBox "Component added successfully"

    Unload Me

End Sub
```

Figure C-13: VBA Form for adding components

Input Template.xlsm - AddProcessForm (Code)

ComboBox1 | DropButtonClick

```vba
    Set ws = ThisWorkbook.Sheets("Node Keys")
    Set searchRange = ws.Range("A2:A10000")
    Set foundCell = searchRange.Find(comboBoxValue, _
        LookIn:=xlValues, lookat:=xlWhole)

    nodeIndex = foundCell.Row - 1

    Set ws = ThisWorkbook.Sheets("Process Keys")
    lastRow = ws.Cells(ws.Rows.Count, nodeIndex).End(xlUp).Row

    'Check if Process already exists
    Set searchRange = ws.Range(ws.Cells(1, nodeIndex).Address(), ws.Cells(lastRow, nodeIndex).Address())
    For Each cell In searchRange
        If cell.Value = textBoxValue Then
            MsgBox "That Process already exists within the Node"
            Exit Sub
        End If
    Next cell

    'Add Process under Node and sort alphabetically
    ws.Cells(lastRow + 1, nodeIndex).Value = textBoxValue

    Set searchRange = ws.Range(ws.Cells(2, nodeIndex).Address(), ws.Cells(lastRow + 1, nodeIndex).Address(
    ws.Sort.SortFields.Clear
    ws.Sort.SortFields.Add2 Key:=searchRange, _
        SortOn:=xlSortOnValues, Order:=xlAscending, DataOption:= _
        xlSortNormal
    With ws.Sort
        .SetRange searchRange
        .Header = xlNo
        .MatchCase = False
        .Orientation = xlTopToBottom
        .SortMethod = xlPinYin
        .Apply
    End With

    MsgBox "Process added successfully"

    Unload Me

End Sub
```

Input Template.xlsm - AddProcessForm (Code)

ComboBox1 | DropButtonClick

```vba
Private Sub ComboBox1_DropButtonClick()
    Dim ws As Worksheet
    Dim lastRow As Long

    Set ws = ThisWorkbook.Sheets("Node Keys")
    lastRow = ws.Cells(ws.Rows.Count, "A").End(xlUp).Row
    Set searchRange = ws.Range("A2:A" & lastRow)
    ComboBox1.List = searchRange.Value
End Sub

Private Sub CommandButton1_Click()
    Dim ws As Worksheet
    Dim targetSheet As Worksheet
    Dim lastRow As Long
    Dim textBoxValue As String
    Dim comboBoxValue As String
    Dim cell As Range
    Dim foundCell As Range
    Dim nodeIndex As Long

    textBoxValue = Me.TextBox1.Value
    comboBoxValue = Me.ComboBox1.Value

    'Check if a Node has been selected
    If comboBoxValue = "" Then
        MsgBox "Please select a Node first"
        Exit Sub
    End If

    'Check if the new Process has a name
    If textBoxValue = "" Then
        MsgBox "Please name the Process you want to add first"
        Exit Sub
    End If

    'Find the Processes of the selected Node
    Set ws = ThisWorkbook.Sheets("Node Keys")
    Set searchRange = ws.Range("A2:A10000")
    Set foundCell = searchRange.Find(comboBoxValue, _
        LookIn:=xlValues, lookat:=xlWhole)

    nodeIndex = foundCell.Row - 1
```

Figure C-14: VBA Form for adding processes

Input Template.xlsm - ComponentActionsForm (Code)

AddComponentButton | Click

```
Private Sub AddComponentButton_Click()
    AddComponentForm.Show
End Sub

Private Sub ExitNodeFormButton_Click()
    Unload Me
End Sub

Private Sub RemoveComponentButton_Click()
    RemoveComponentForm.Show
End Sub
```

Figure C-15: VBA Form for creating component actions

Input Template.xlsm - NodeActionsForm (Code)

AddNodeButton | Click

```
Private Sub AddNodeButton_Click()
    AddNodeForm.Show
End Sub

Private Sub ExitNodeFormButton_Click()
    Unload Me
End Sub

Private Sub RemoveNodeButton_Click()
    RemoveNodeForm.Show
End Sub
```

Figure C-16: VBA Form for creating node actions

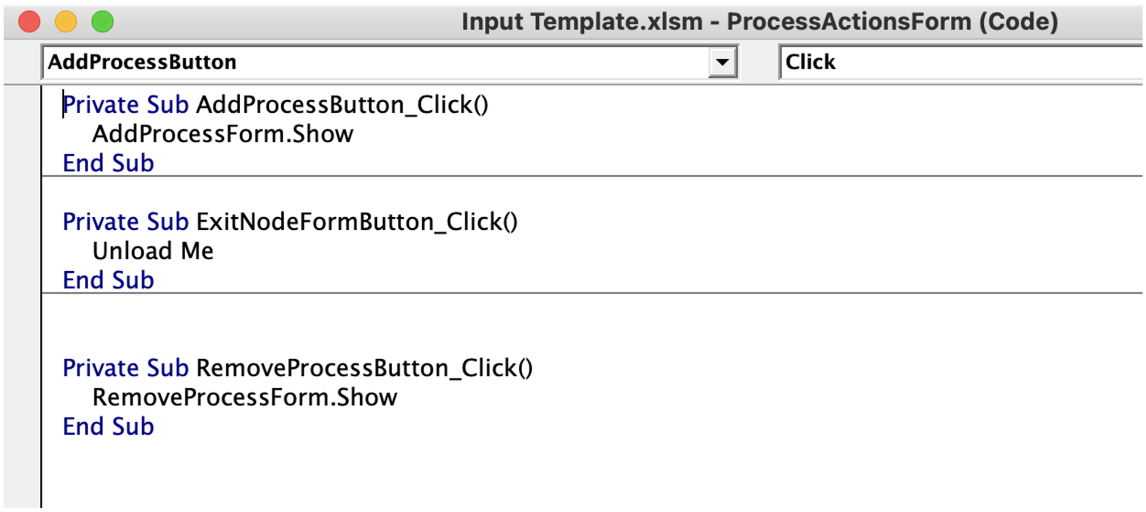


Figure C-17: VBA Form for creating process actions

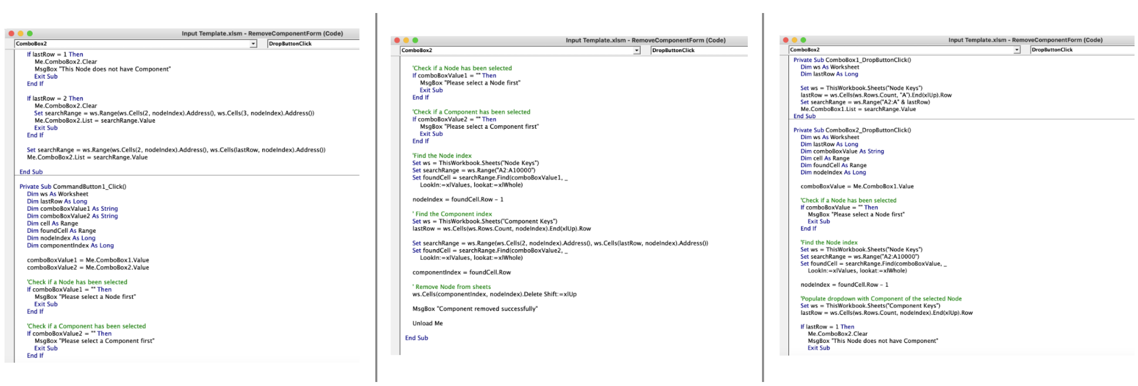
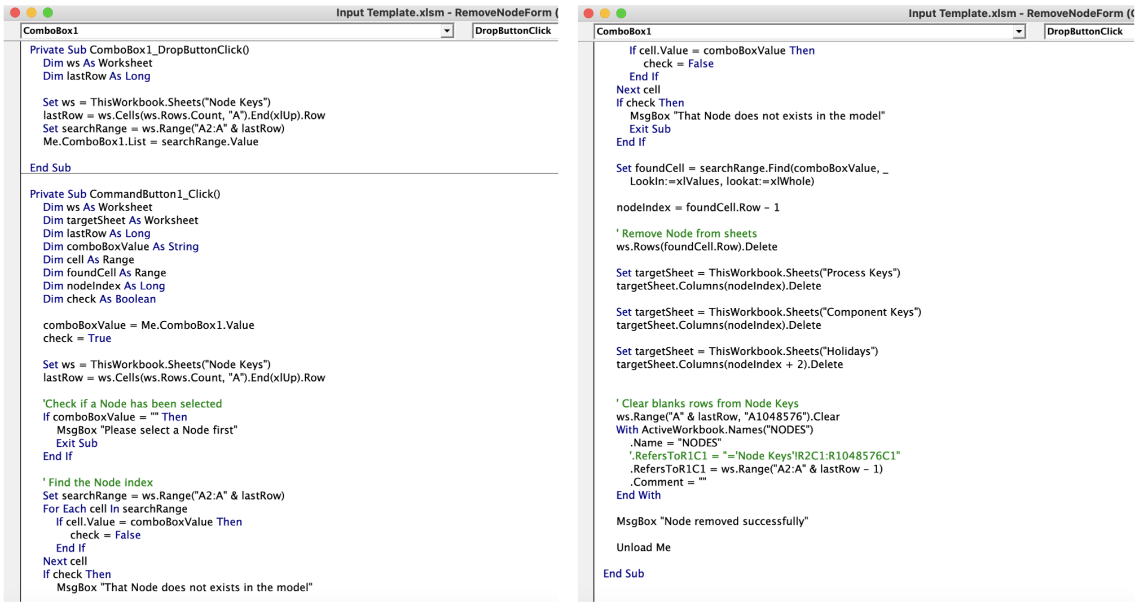


Figure C-18: VBA Form for removing components



```
Input Template.xlsm - RemoveNodeForm (
ComboBox1    DropButtonClick

Private Sub ComboBox1_DropButtonClick()
    Dim ws As Worksheet
    Dim lastRow As Long

    Set ws = ThisWorkbook.Sheets("Node Keys")
    lastRow = ws.Cells(ws.Rows.Count, "A").End(xlUp).Row
    Set searchRange = ws.Range("A2:A" & lastRow)
    Me.ComboBox1.List = searchRange.Value

End Sub

Private Sub CommandButton1_Click()
    Dim ws As Worksheet
    Dim targetSheet As Worksheet
    Dim lastRow As Long
    Dim comboBoxValue As String
    Dim cell As Range
    Dim foundCell As Range
    Dim nodeIndex As Long
    Dim check As Boolean

    comboBoxValue = Me.ComboBox1.Value
    check = True

    Set ws = ThisWorkbook.Sheets("Node Keys")
    lastRow = ws.Cells(ws.Rows.Count, "A").End(xlUp).Row

    'Check if a Node has been selected
    If comboBoxValue = "" Then
        MsgBox "Please select a Node first"
        Exit Sub
    End If

    ' Find the Node index
    Set searchRange = ws.Range("A2:A" & lastRow)
    For Each cell In searchRange
        If cell.Value = comboBoxValue Then
            check = False
        End If
    Next cell
    If check Then
        MsgBox "That Node does not exists in the model"
```

```
Input Template.xlsm - RemoveNodeForm (C
ComboBox1    DropButtonClick

        If cell.Value = comboBoxValue Then
            check = False
        End If
    Next cell
    If check Then
        MsgBox "That Node does not exists in the model"
        Exit Sub
    End If

    Set foundCell = searchRange.Find(comboBoxValue, _
        LookIn:=xlValues, lookat:=xlWhole)

    nodeIndex = foundCell.Row - 1

    ' Remove Node from sheets
    ws.Rows(foundCell.Row).Delete

    Set targetSheet = ThisWorkbook.Sheets("Process Keys")
    targetSheet.Columns(nodeIndex).Delete

    Set targetSheet = ThisWorkbook.Sheets("Component Keys")
    targetSheet.Columns(nodeIndex).Delete

    Set targetSheet = ThisWorkbook.Sheets("Holidays")
    targetSheet.Columns(nodeIndex + 2).Delete


    ' Clear blanks rows from Node Keys
    ws.Range("A" & lastRow, "A1048576").Clear
    With ActiveWorkbook.Names("NODES")
        .Name = "NODES"
        '.RefersToR1C1 = "='Node Keys'!R2C1:R1048576C1"
        .RefersToR1C1 = ws.Range("A2:A" & lastRow - 1)
        .Comment = ""
    End With

    MsgBox "Node removed successfully"

    Unload Me

End Sub
```

Figure C-19: VBA Form for removing nodes

```
Input Template.xlsm - RemoveProcessForm (Code)
ComboBox1    DropButtonClick

Private Sub ComboBox1_DropButtonClick()
    Dim ws As Worksheet
    Dim lastRow As Long

    Set ws = ThisWorkbook.Sheets("Node Keys")
    lastRow = ws.Cells(ws.Rows.Count, "A").End(xlUp).Row
    Set searchRange = ws.Range("A2:A" & lastRow)
    Me.ComboBox1.List = searchRange.Value
End Sub

Private Sub ComboBox2_DropButtonClick()
    Dim ws As Worksheet
    Dim lastRow As Long
    Dim comboBoxValue As String
    Dim cell As Range
    Dim foundCell As Range
    Dim nodeIndex As Long

    comboBoxValue = Me.ComboBox1.Value

    'Check if a Node has been selected
    If comboBoxValue = "" Then
        MsgBox "Please select a Node first"
        Exit Sub
    End If

    'Find the Node index
    Set ws = ThisWorkbook.Sheets("Node Keys")
    Set searchRange = ws.Range("A2:A10000")
    Set foundCell = searchRange.Find(comboBoxValue, _
        LookIn:=xlValues, lookat:=xlWhole)

    nodeIndex = foundCell.Row - 1

    'Populate dropdown with Processes of the selected Node
    Set ws = ThisWorkbook.Sheets("Process Keys")
    lastRow = ws.Cells(ws.Rows.Count, nodeIndex).End(xlUp).Row

    If lastRow = 1 Then
        Me.ComboBox2.Clear
        MsgBox "This Node does not have Processes"
        Exit Sub
```

```
Input Template.xlsm - RemoveProcessForm (Code)
ComboBox1    DropButtonClick

    comboBoxValue1 = Me.ComboBox1.Value
    comboBoxValue2 = Me.ComboBox2.Value

    'Check if a Node has been selected
    If comboBoxValue1 = "" Then
        MsgBox "Please select a Node first"
        Exit Sub
    End If

    'Check if a Process has been selected
    If comboBoxValue2 = "" Then
        MsgBox "Please select a Process first"
        Exit Sub
    End If

    'Find the Node index
    Set ws = ThisWorkbook.Sheets("Node Keys")
    Set searchRange = ws.Range("A2:A10000")
    Set foundCell = searchRange.Find(comboBoxValue1, _
        LookIn:=xlValues, lookat:=xlWhole)

    nodeIndex = foundCell.Row - 1

    ' Find the Process index
    Set ws = ThisWorkbook.Sheets("Process Keys")
    lastRow = ws.Cells(ws.Rows.Count, nodeIndex).End(xlUp).Row

    Set searchRange = ws.Range(ws.Cells(2, nodeIndex).Address(), ws.Cells(lastRow, nodeIndex).Address())
    Set foundCell = searchRange.Find(comboBoxValue2, _
        LookIn:=xlValues, lookat:=xlWhole)

    processIndex = foundCell.Row

    ' Remove Node from sheets
    ws.Cells(processIndex, nodeIndex).Delete Shift:=xlUp

    MsgBox "Process removed successfully"

    Unload Me

End Sub
```

```
Input Template.xlsm - RemoveProcessForm (Code)
ComboBox1    DropButtonClick

    If lastRow = 1 Then
        Me.ComboBox2.Clear
        MsgBox "This Node does not have Processes"
        Exit Sub
    End If

    If lastRow = 2 Then
        Me.ComboBox2.Clear
        Set searchRange = ws.Range(ws.Cells(2, nodeIndex).Address(), ws.Cells(3, nodeIndex).Address())
        Me.ComboBox2.List = searchRange.Value
        Exit Sub
    End If

    Set searchRange = ws.Range(ws.Cells(2, nodeIndex).Address(), ws.Cells(lastRow, nodeIndex).Address())
    Me.ComboBox2.List = searchRange.Value

End Sub

Private Sub CommandButton1_Click()
    Dim ws As Worksheet
    Dim lastRow As Long
    Dim comboBoxValue1 As String
    Dim comboBoxValue2 As String
    Dim cell As Range
    Dim foundCell As Range
    Dim nodeIndex As Long
    Dim processIndex As Long

    comboBoxValue1 = Me.ComboBox1.Value
    comboBoxValue2 = Me.ComboBox2.Value

    'Check if a Node has been selected
    If comboBoxValue1 = "" Then
        MsgBox "Please select a Node first"
        Exit Sub
    End If

    'Check if a Process has been selected
    If comboBoxValue2 = "" Then
        MsgBox "Please select a Process first"
        Exit Sub
    End If
```

Figure C-20: VBA Form for removing processes

C.3 Data Visualization - Dashboard

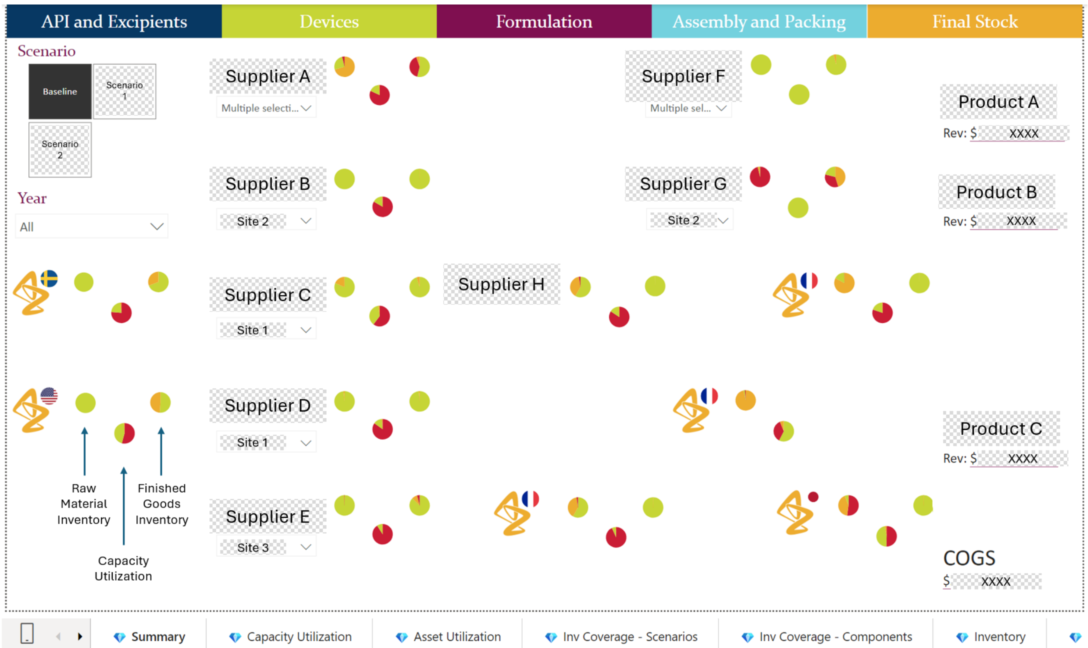


Figure C-21: Summary Tab

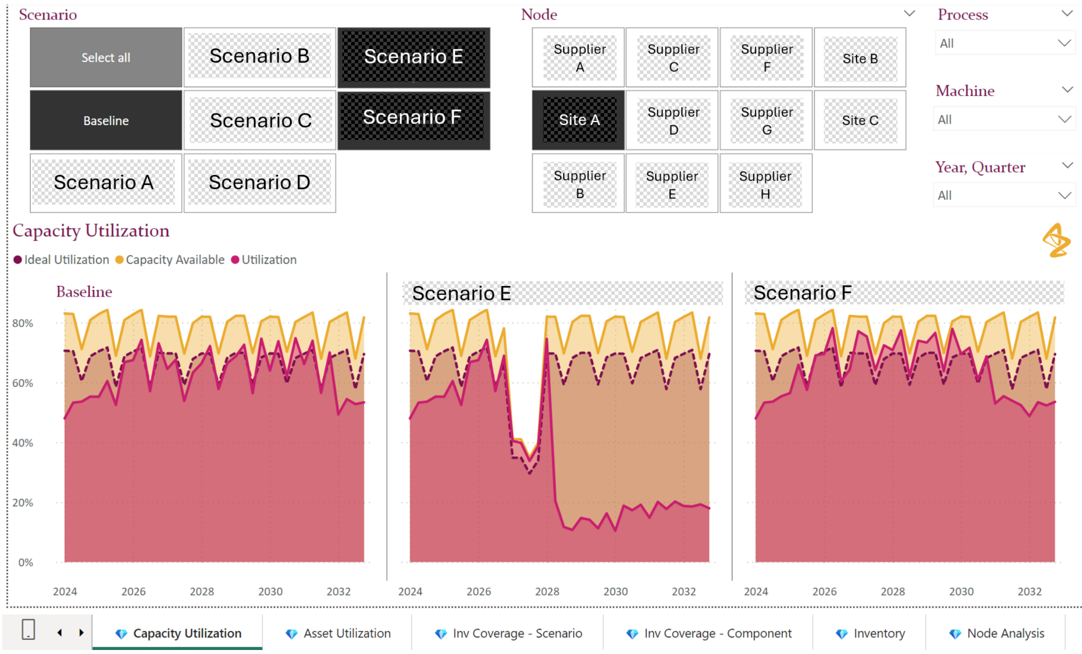


Figure C-22: Capacity Utilization Tab

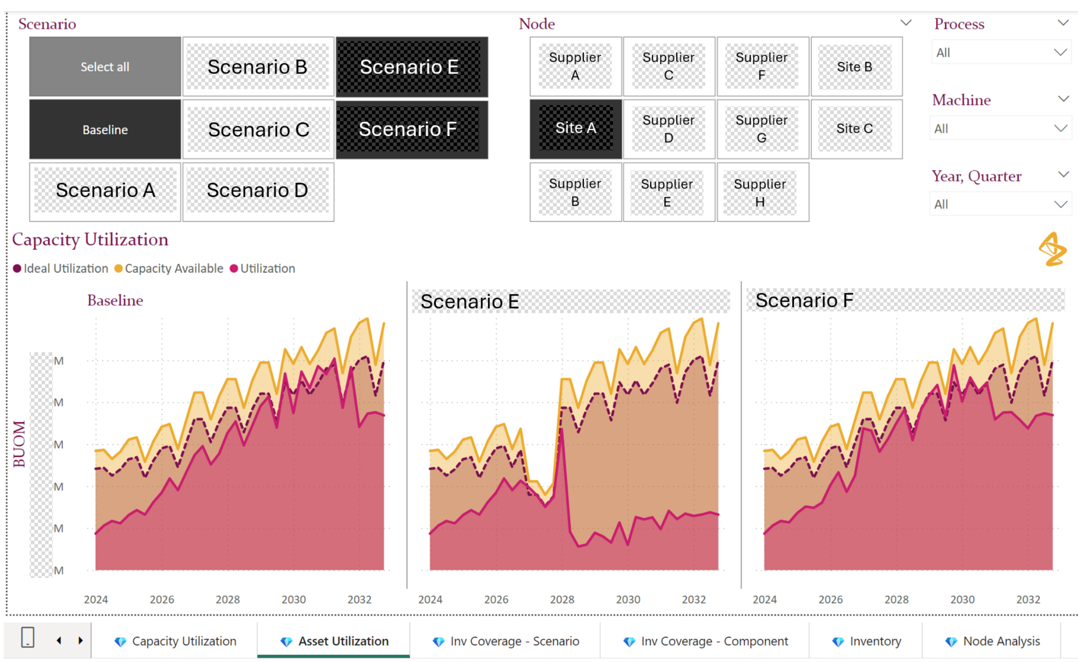


Figure C-23: Asset Utilization Tab

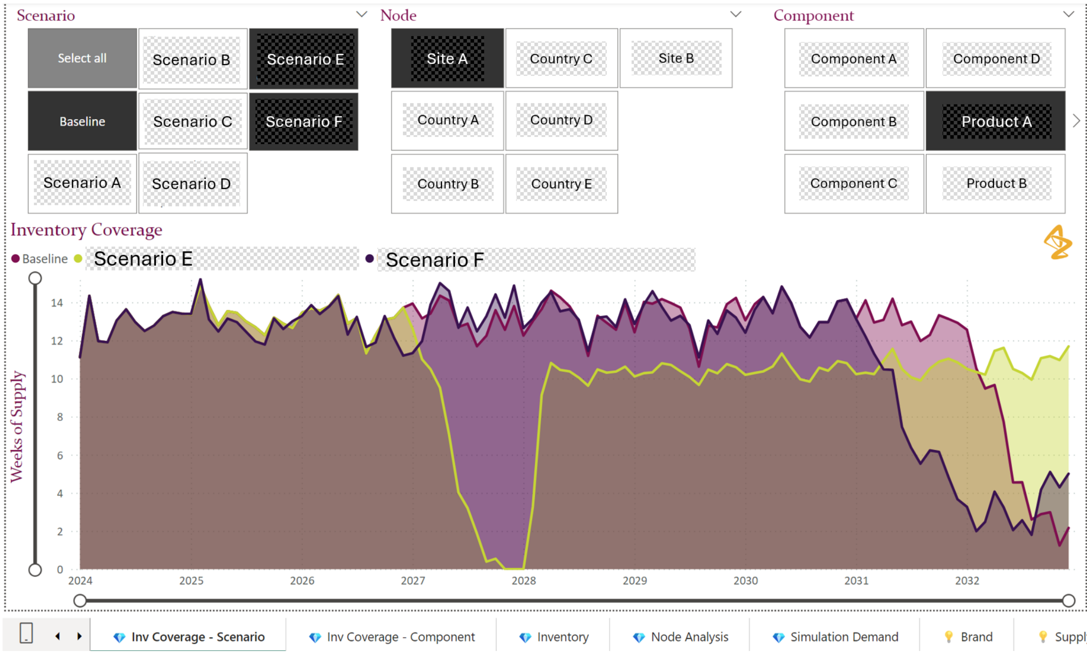


Figure C-24: Inventory Coverage Scenarios Tab

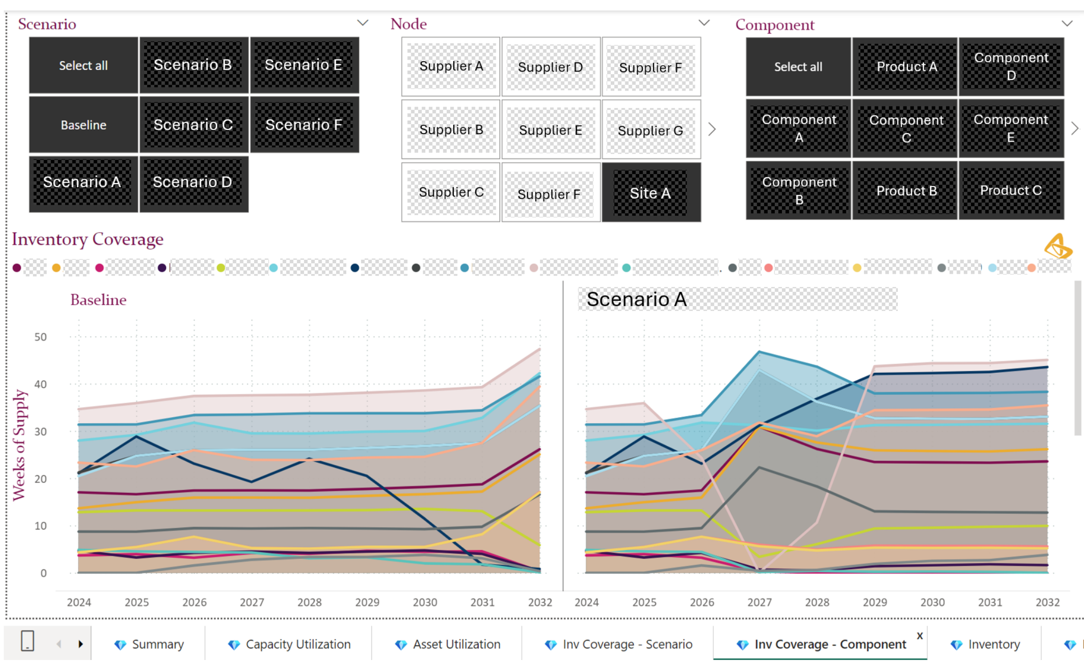


Figure C-25: Inventory Coverage Components Tab

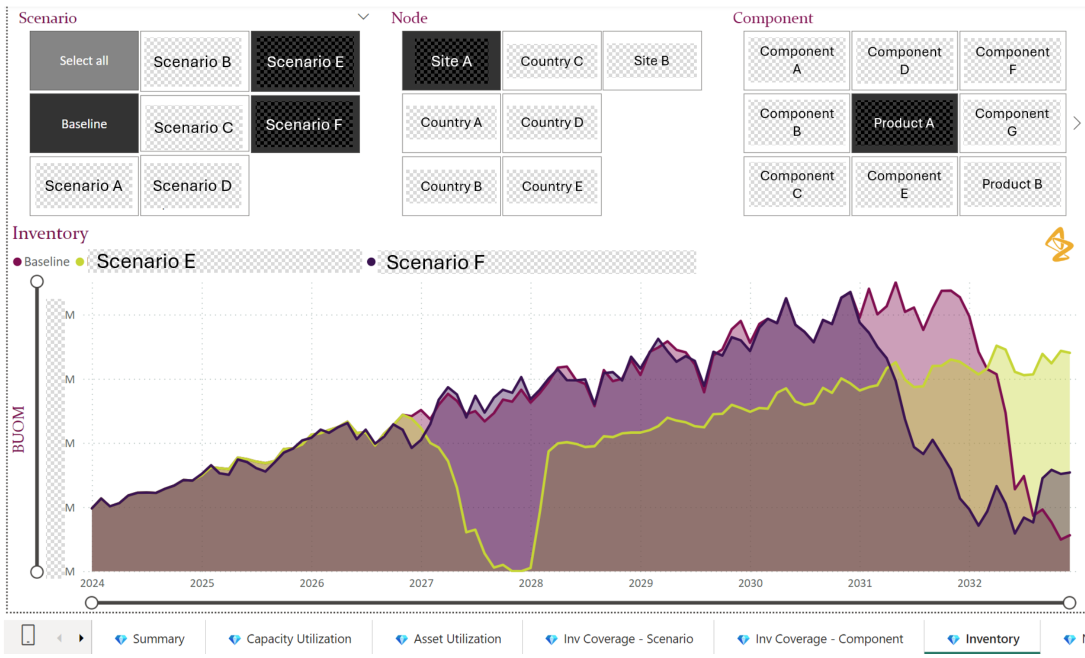


Figure C-26: Inventory Tab

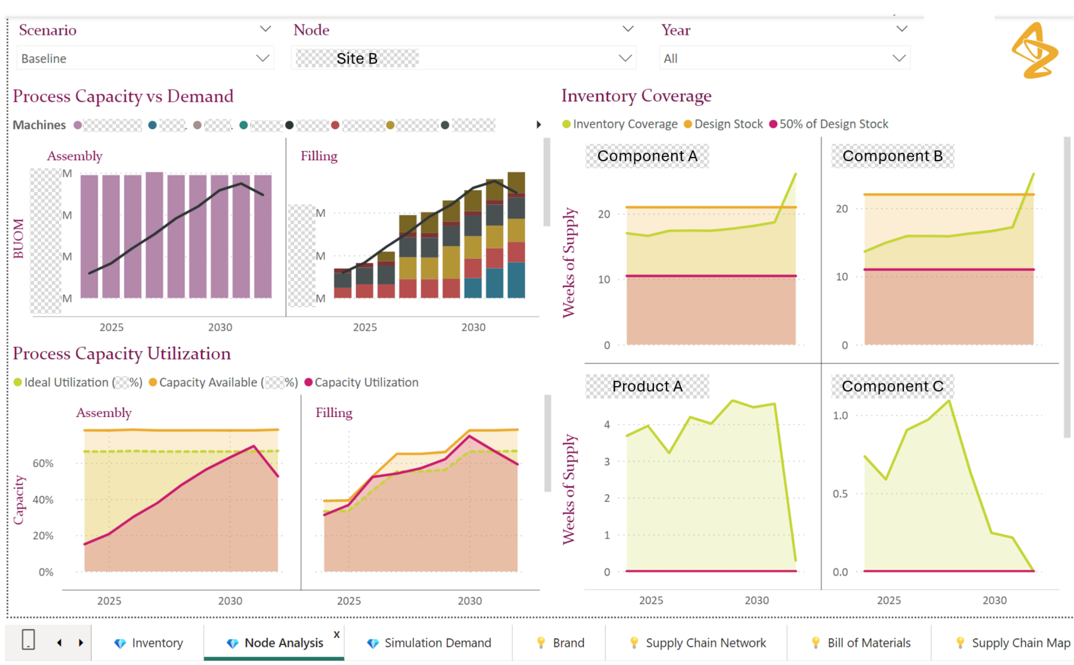


Figure C-27: Node Analysis Tab

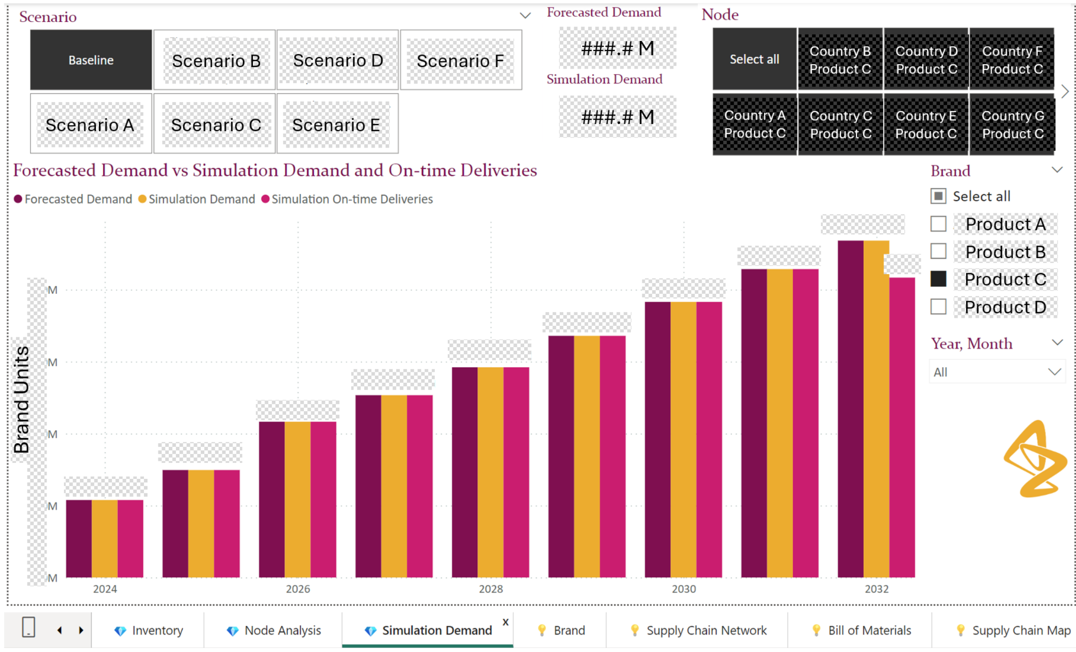


Figure C-28: Simulation Demand Tab

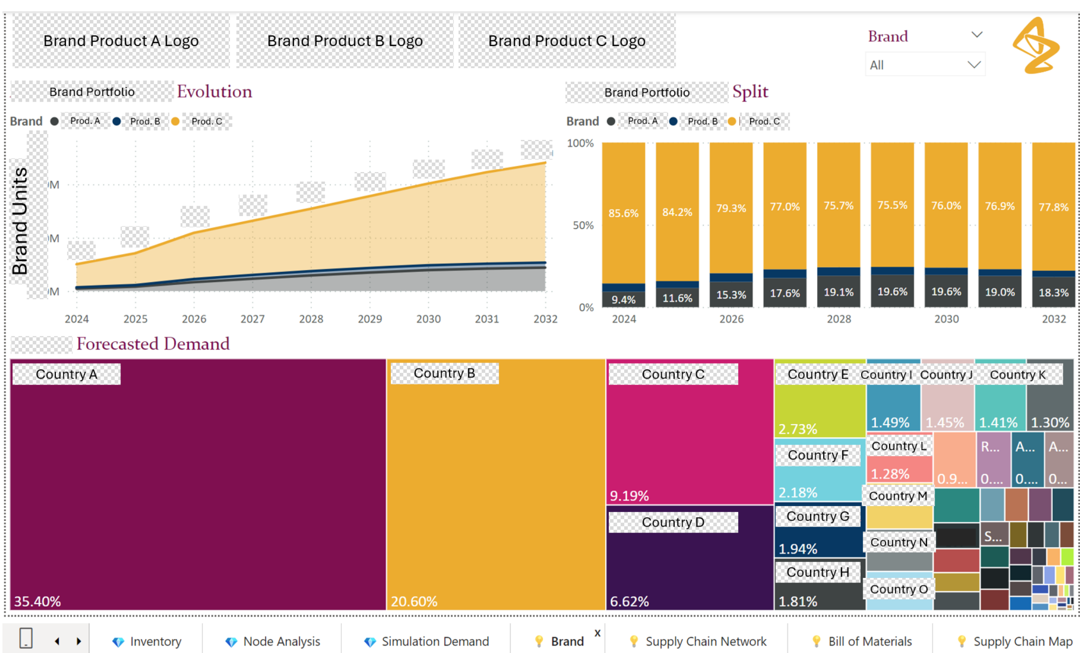


Figure C-29: Brand Tab

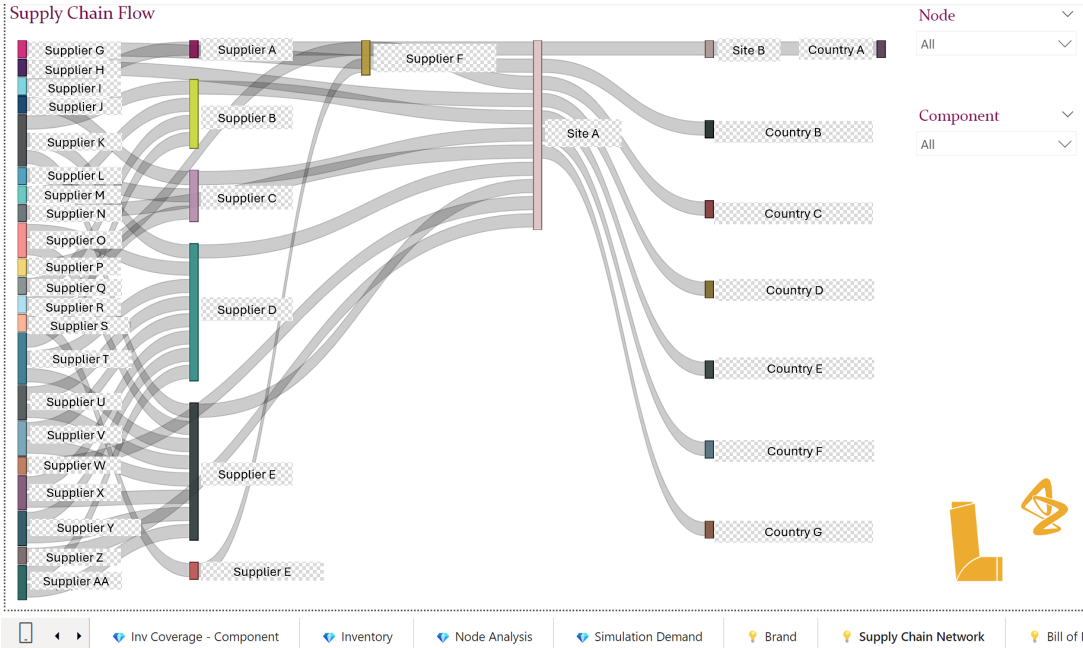


Figure C-30: Supply Chain Network Tab

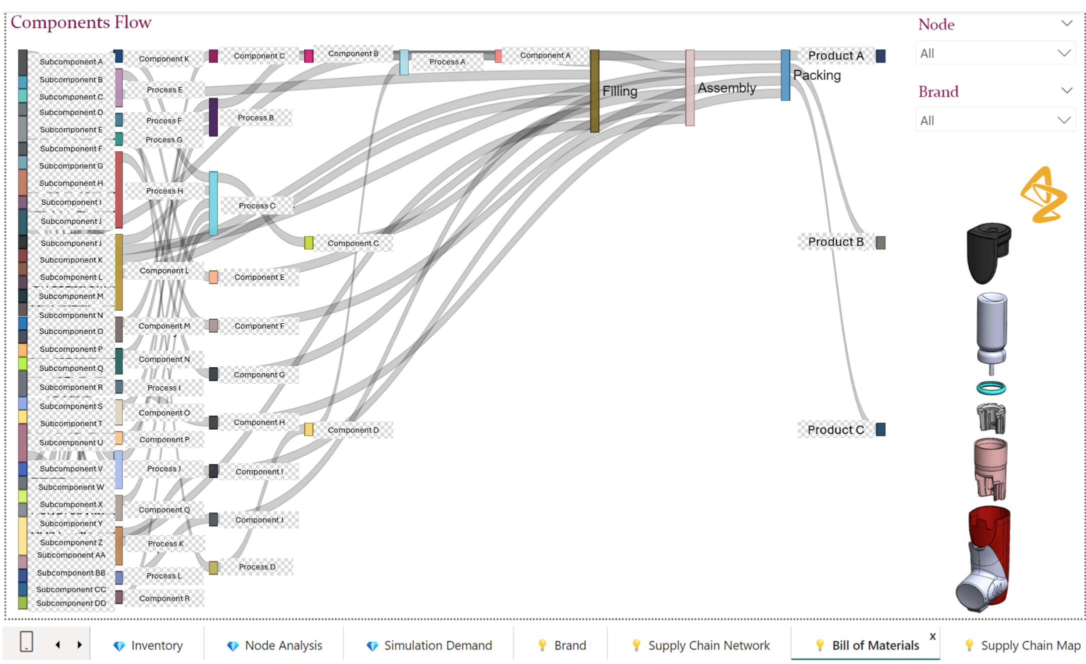


Figure C-31: Bill of Materials Tab

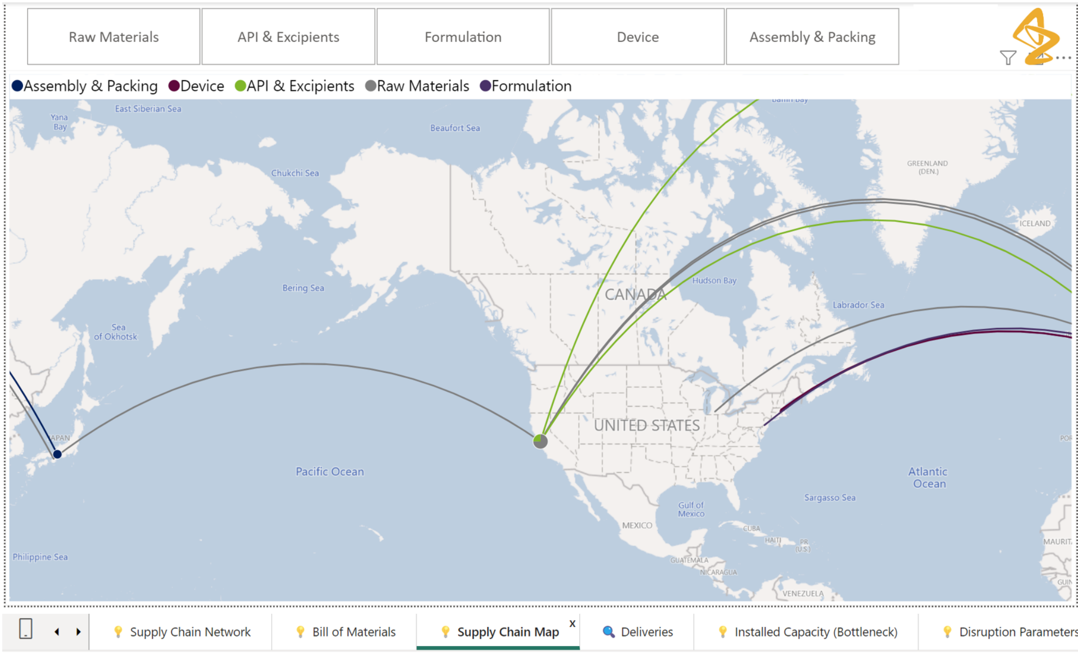


Figure C-32: Supply Chain Map Tab

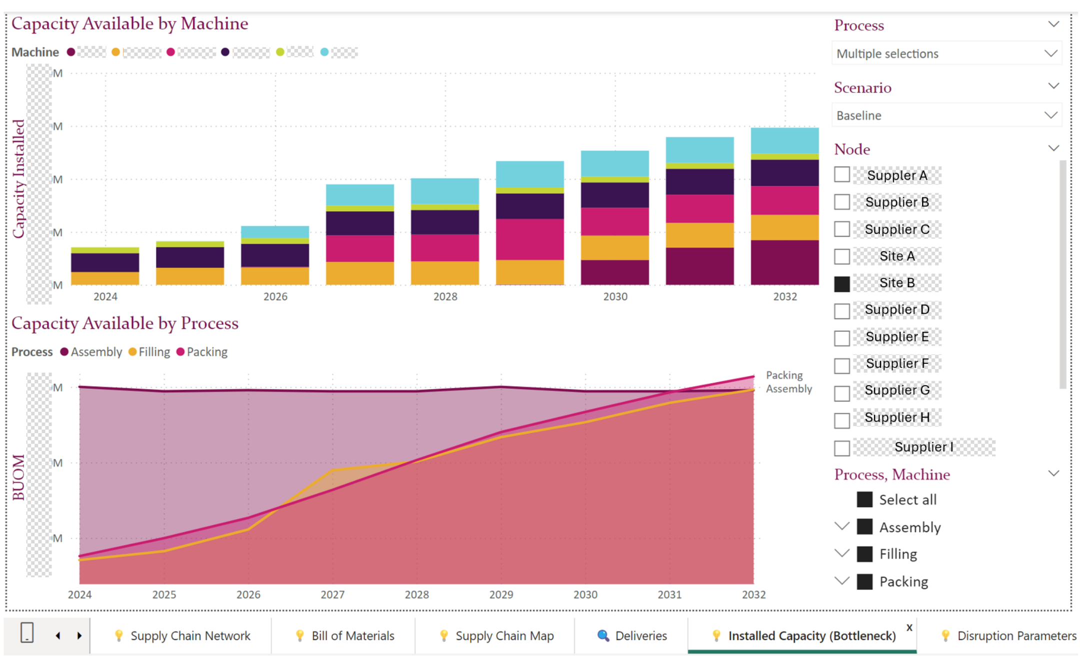


Figure C-33: Installed Capacity Tab

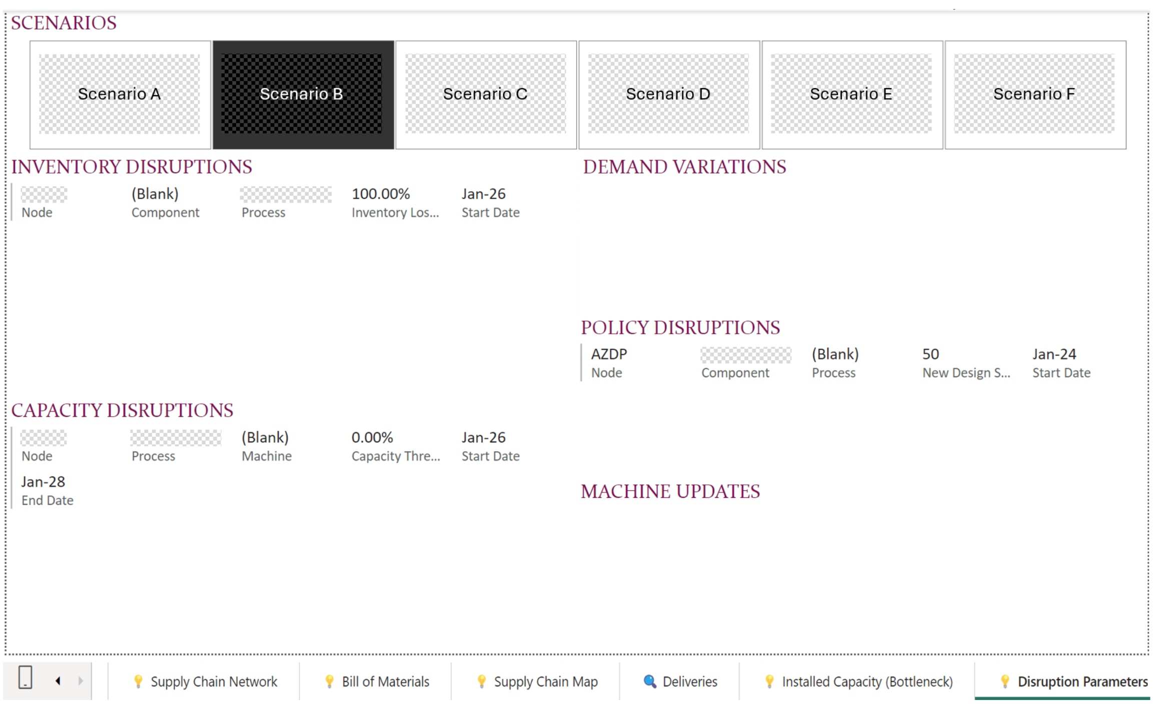


Figure C-34: Disruption Parameters Tab

THIS PAGE INTENTIONALLY LEFT BLANK

Bibliography

[1] AstraZeneca. *AstraZeneca Annual Report and Form 20-F Information 2023*. 2024. URL: http://www.astrazeneca.com/annualreport2023 (visited on 02/27/2024).

[2] AstraZeneca. "Full Year and Q42023 Results". Feb. 8, 2024. URL: https://www.astrazeneca.com/investor-relations/full-year-and-q4-2023-results-event.html (visited on 02/11/2024).

[3] AstraZeneca. *Our Company*. URL: https://www.astrazeneca.com/our-company.html (visited on 11/11/2023).

[4] AstraZeneca. *AstraZeneca's Sustainability Partner Guide and Framework*. URL: https://www.astrazeneca.com/content/dam/az/PDF/Sustainability/2019/Sustainability%20Toolkit%20and%20Framework_A4_Digital_191218.pdf (visited on 02/20/2024).

[5] AstraZeneca. *'Device A1' approved by the US FDA for patients with COPD*. Apr. 25, 2016. URL: https://www.astrazeneca.com/media-centre/press-releases/2016/bevespi-aerosphere-approved-by-the-us-fda-for-patients-with-copd-25042016.html (visited on 06/15/2023).

[6] Adrian Kemp. *'Device A2' approved in the US for the maintenance treatment of COPD*. July 24, 2020. URL: https://www.astrazeneca.com/media-centre/press-releases/2020/breztri-aerosphere-approved-in-the-us-for-copd.html (visited on 06/16/2023).

[7] Adrian Kemp. *'Device A3' approved in the US for asthma*. Jan. 11, 2023. URL: https://www.astrazeneca.com/media-centre/press-releases/2023/airsupra-pt027-approved-in-the-us-for-asthma.html (visited on 06/21/2023).

[8] AstraZeneca. *AstraZeneca Development Pipeline as at 8 February 2024*. URL: https://www.astrazeneca.com/content/dam/az/Investor_Relations/annual-report-2023/pdf/AstraZeneca_Development_Pipeline_2023.pdf (visited on 02/23/2024).

[9] AstraZeneca. *AstraZeneca announces initiation of THARROS – a Phase III clinical trial investigating the potential of 'Device A2' to improve cardiopulmonary outcomes in people with COPD*. Mar. 13, 2024. URL: https://www.astrazeneca.com/media-centre/medical-releases/astrazeneca-announces-initiation-tharros-phase-iii-clinical-trial-investigating-

potential-breztri-improve-cardiopulmonary-outcomes-people-with-copd.html (visited on 04/30/2024).

[10] AstraZeneca. 阿斯利康加码在华投资，建设青岛生产供应基地并设立区域总部*[Translated to English]*. June 20, 2022. URL: https://www.astrazeneca.com.cn/zh/media/press-releases/2022/_.html (visited on 02/02/2024).

[11] AstraZeneca. *AstraZeneca progresses Ambition Zero Carbon programme with Honeywell partnership to develop next-generation respiratory inhalers*. Feb. 22, 2022. URL: https://www.astrazeneca.com/media-centre/press-releases/2022/astrazeneca-progresses-ambition-zero-carbon-programme-with-honeywell-partnership-to-develop-next-generation-respiratory-inhalers.html (visited on 12/19/2023).

[12] S. P. Newman. "AEROSOLS". In: *Encyclopedia of Respiratory Medicine*. Ed. by Geoffrey J. Laurent and Steven D. Shapiro. Oxford: Academic Press, 2006, pp. 58–64. ISBN: 978-0-12-370879-3. DOI: https://doi.org/10.1016/B0-12-370879-6/00019-3. URL: https://www.sciencedirect.com/science/article/pii/B0123708796000193.

[13] Gianluca Benigno et al. *The GSCPI: A New Barometer of Global Supply Chain Pressures*. 1017. May 2022. URL: https://www.newyorkfed.org/medialibrary/media/research/staff_reports/sr1017.pdf?sc_lang=en (visited on 02/05/2024).

[14] Council of Economic Advisers. *Issue Brief: Supply Chain Resilience*. The White House. Nov. 30, 2023. URL: https://www.whitehouse.gov/cea/written-materials/2023/11/30/issue-brief-supply-chain-resilience/ (visited on 02/05/2024).

[15] HIDEAKI RYUGEN. *TSMC speeds diversification push with new Japan chip plant*. Nikkei Asia. Feb. 8, 2024. URL: https://asia.nikkei.com/Business/Tech/Semiconductors/TSMC-speeds-diversification-push-with-new-Japan-chip-plant (visited on 02/25/2024).

[16] The White House. *FACT SHEET: CHIPS and Science Act Will Lower Costs, Create Jobs, Strengthen Supply Chains, and Counter China*. The White House. Aug. 9, 2022. URL: https://www.whitehouse.gov/briefing-room/statements-releases/2022/08/09/fact-sheet-chips-and-science-act-will-lower-costs-create-jobs-strengthen-supply-chains-and-counter-china/ (visited on 11/25/2023).

[17] Sandor Boyson et al. "How Exposed Is Your Supply Chain to Climate Risks?" In: *Harvard Business Review* (May 2, 2022). ISSN: 0017-8012. URL: https://hbr.org/2022/05/how-exposed-is-your-supply-chain-to-climate-risks (visited on 01/25/2024).

[18] Lawson Brigham. *The Suez Canal and Global Trade Routes*. U.S. Naval Institute. May 1, 2021. URL: https://www.usni.org/magazines/proceedings/2021/may/suez-canal-and-global-trade-routes (visited on 01/22/2024).

[19] Costas Paris. *Two Canals, Two Big Problems—One Global Shipping Mess*. WSJ. Section: Business. Mar. 10, 2024. URL: https://www.wsj.com/business/logistics/shipping-panama-red-sea-suez-canal-edc91172 (visited on 03/22/2024).

[20] Knut Alicke, Richa Gupta, and Vera Trautwein. *Resetting supply chains for the next normal | McKinsey*. July 21, 2020. URL: https://www.mckinsey.com/capabilities/operations/our-insights/resetting-supply-chains-for-the-next-normal (visited on 01/05/2024).

[21] Knut Alicke et al. *Tech and regionalization bolster supply chains, but complacency looms | McKinsey*. Nov. 3, 2023. URL: https://www.mckinsey.com/capabilities/operations/our-insights/tech-and-regionalization-bolster-supply-chains-but-complacency-looms (visited on 02/02/2024).

[22] AstraZeneca. *Data Science & Artificial Intelligence: Unlocking new science insights*. Sept. 2022. URL: https://www.astrazeneca.com/r-d/data-science-and-ai.html (visited on 01/25/2024).

[23] AstraZeneca. *AstraZeneca Annual Report and Form 20-F Information 2022*. 2023. URL: www.astrazeneca.com/annualreport2022 (visited on 01/13/2024).

[24] AstraZeneca. *AstraZeneca Annual Report and Form 20-F Information 2020*. 2021. URL: www.astrazeneca.com/annualreport2020 (visited on 01/13/2024).

[25] AstraZeneca. *Developing the next generation of drug delivery technologies*. July 12, 2023. URL: https://www.astrazeneca.com/what-science-can-do/topics/next-generation-therapeutics/developing-the-next-generation-of-drug-delivery-technologies.html (visited on 08/12/2023).

[26] AstraZeneca. *Smart factories - delivering more for patients worldwide*. Aug. 25, 2022. URL: https://www.astrazeneca.com/what-science-can-do/topics/technologies/smart-factories-delivering-more-for-patients-worldwide.html (visited on 10/24/2023).

[27] AstraZeneca. *AstraZeneca launches Evinova, a health-tech business to accelerate innovation across the life sciences sector, the delivery of clinical trials and better health outcomes*. Nov. 20, 2023. URL: https://www.astrazeneca.com/media-centre/press-releases/2023/astrazeneca-launches-evinova-health-tech-business-to-accelerate-innovation-across-the-life-sciences-sector.html (visited on 12/14/2023).

[28] Serhiy Y. Ponomarov and Mary C. Holcomb. "Understanding the concept of supply chain resilience". In: *The International Journal of Logistics Management* 20.1 (Jan. 1, 2009). Publisher: Emerald Group Publishing Limited, pp. 124–143. ISSN: 0957-4093. DOI: 10.1108/09574090910954873. URL: https://doi.org/10.1108/09574090910954873 (visited on 08/28/2023).

[29] Yossi Sheffi and James B. Rice Jr. "A Supply Chain View of the Resilient Enterprise". In: *MIT Sloan Management Review* (Fall 2005 Oct. 15, 2005). URL: https://sloanreview.mit.edu/article/a-supply-chain-view-of-the-resilient-enterprise/ (visited on 09/28/2023).

[30] David Simchi-Levi, He Wang, and Yehua Wei. "Increasing Supply Chain Robustness through Process Flexibility and Inventory". In: *Production and Operations Management* 27.8 (2018), pp. 1476–1491. ISSN: 1937-5956. DOI: 10.1111/poms.12887. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/poms.12887 (visited on 07/26/2023).

[31] Sunil Chopra and ManMohan S. Sodhi. "Reducing the Risk of Supply Chain Disruptions". In: *MIT Sloan Management Review* (Mar. 18, 2014). URL: https://sloanreview.mit.edu/article/reducing-the-risk-of-supply-chain-disruptions/ (visited on 09/08/2023).

[32] Sunil Chopra and ManMohan S. Sodhi. "Managing Risk to Avoid Supply-Chain Breakdown". In: *MIT Sloan Management Review* (Oct. 15, 2004). URL: https://sloanreview.mit.edu/article/managing-risk-to-avoid-supplychain-breakdown/ (visited on 09/08/2023).

[33] David Simchi-Levi et al. "Identifying Risks and Mitigating Disruptions in the Automotive Supply Chain". In: *Prof. Simchi-Levi via Anne Graham* (Oct. 2015). Accepted: 2016-03-24T23:41:10Z Publisher: Institute for Operations Research and the Management Sciences (INFORMS). ISSN: 0092-2102. URL: https://dspace.mit.edu/handle/1721.1/101782 (visited on 07/28/2023).

[34] David Simchi-Levi, William Schmidt, and Yehua Wei. "From Superstorms to Factory Fires: Managing Unpredictable Supply-Chain Disruptions". In: *Harvard Business Review* (January–February 2014 2014). URL: https://hbr.org/2014/01/from-superstorms-to-factory-fires-managing-unpredictable-supply-chain-disruptions (visited on 07/12/2023).

[35] David Simchi-Levi. "Find the Weak Link in Your Supply Chain". In: *Harvard Business Review* (June 9, 2015). Section: Operations strategy. ISSN: 0017-8012. URL: https://hbr.org/2015/06/find-the-weak-link-in-your-supply-chain (visited on 07/13/2023).

[36] David Simchi-Levi. "A New Approach to Manage Supply Chain Risk". In: *Harvard Business Review* (Oct. 21, 2015). Section: Operations and supply chain management. ISSN: 0017-8012. URL: https://hbr.org/webinar/2015/11/a-new-approach-to-manage-supply-chain-risk (visited on 07/07/2023).

[37] Michael W. Grieves. "Digital Twins: Past, Present, and Future". In: *The Digital Twin*. Ed. by Noel Crespi, Adam T. Drobot, and Roberto Minerva. Cham: Springer International Publishing, 2023, pp. 97–121. ISBN: 978-3-031-21343-4. DOI: 10.1007/978-3-031-21343-4_4. URL: https://doi.org/10.1007/978-3-031-21343-4_4 (visited on 09/23/2023).

[38] Eric VanDerHorn and Sankaran Mahadevan. "Digital Twin: Generalization, characterization and implementation". In: *Decision Support Systems* 145 (June 1, 2021), p. 113524. ISSN: 0167-9236. DOI: 10.1016/j.dss.2021.113524. URL: https://www.sciencedirect.com/science/article/pii/S0167923621000348 (visited on 11/20/2023).

[39] Özden Tozanli and María Jesús Sáenz. "Unlocking the Potential of Digital Twins in Supply Chains". In: *MIT Sloan Management Review* (Aug. 18, 2022). URL: https://sloanreview.mit.edu/article/unlocking-the-potential-of-digital-twins-in-supply-chains/ (visited on 07/23/2023).

[40] Özden Tozanli and María Jesús Sáenz. "Four Misconceptions are Hampering the Advancement of Digital Twins". In: *Supply Chain Management Review* (July/August 2022 July 7, 2022), pp. 10–13. URL: https://www.scmr.com/article/four_misconceptions_are_hampering_the_advancement_of_digital_twins (visited on 07/23/2023).

[41] B. Danette Allen. "Digital Twins and Living Models at NASA". NTRS Author Affiliations: Langley Research Center NTRS Meeting Information: Digital Twin Summit; 2021-11-03 to 2021-11-04; undefined NTRS Document ID: 20210023699 NTRS Research Center: Langley Research Center (LaRC). Nov. 3, 2021. URL: https://ntrs.nasa.gov/citations/20210023699 (visited on 01/17/2023).

[42] Jackie Snow. "What Is a Digital Twin? And How Can It Make Companies—and Cities—More Efficient?" In: *Wall Street Journal* (Mar. 17, 2023). ISSN: 0099-9660. URL: https://www.wsj.com/articles/what-is-digital-twin-making-companies-cities-more-efficient-92e551b6 (visited on 01/29/2024).

[43] Rolls-Royce. *Rolls-Royce | Pioneering the IntelligentEngine*. Feb. 5, 2018. URL: https://www.youtube.com/watch?v=9CcbYQ5QA70 (visited on 01/15/2024).

[44] Groupe Renault. *Digital twin of vehicles: when physical and digital models come together*. July 11, 2022. URL: https://www.youtube.com/watch?v=J-edZjYQors (visited on 01/08/2024).

[45] Werner Kritzinger et al. "Digital Twin in manufacturing: A categorical literature review and classification". In: *IFAC-PapersOnLine*. 16th IFAC Symposium on Information Control Problems in Manufacturing INCOM 2018 51.11 (Jan. 1, 2018), pp. 1016–1022. ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2018.08.474. URL: https://www.sciencedirect.com/science/article/pii/S2405896318316021 (visited on 01/08/2024).

[46] Hasso Plattner. *An Introduction to Design Thinking*. URL: https://web.stanford.edu/~mshanks/MichaelShanks/files/509554.pdf (visited on 01/17/2024).

[47] TED. *Tim Brown urges designers to think big*. Sept. 30, 2009. URL: https://www.youtube.com/watch?v=UAinLaT42xY (visited on 02/01/2024).

[48] AstraZeneca. *AstraZeneca Annual Report and Form 20-F Information 2016*. 2017. URL: www.astrazeneca.com/annualreport2016 (visited on 01/13/2024).

[49] AstraZeneca. *AstraZeneca Annual Report and Form 20-F Information 2017*. 2018. URL: www.astrazeneca.com/annualreport2017 (visited on 01/13/2024).

[50] AstraZeneca. *AstraZeneca Annual Report and Form 20-F Information 2018*. 2019. URL: www.astrazeneca.com/annualreport2018 (visited on 01/13/2024).

[51] AstraZeneca. *AstraZeneca Annual Report and Form 20-F Information 2019*. 2020. URL: www.astrazeneca.com/annualreport2019 (visited on 01/13/2024).

[52] AstraZeneca. *AstraZeneca Annual Report and Form 20-F Information 2021*. 2022. URL: www.astrazeneca.com/annualreport2021 (visited on 01/13/2024).

[53] AstraZeneca. *'Device A1' Patient Information*. Mar. 2023. URL: https://den8dhaj6zs0e.cloudfront.net/50fd68b9-106b-4550-b5d0-12b045f8b184/fe60bf8d-1354-42c8-8f3c-5b4ea85ed161/fe60bf8d-1354-42c8-8f3c-5b4ea85ed161_pi_med_guide_rendition__c.pdf (visited on 09/01/2023).

[54] AstraZeneca. *'Device A2' Patient Information*. Jan. 2022. URL: https://den8dhaj6zs0e.cloudfront.net/50fd68b9-106b-4550-b5d0-12b045f8b184/9d44f9af-438a-448b-bb5c-dae506e17e49/9d44f9af-438a-448b-bb5c-dae506e17e49_pi_med_guide_rendition__c.pdf (visited on 09/01/2023).

[55] AstraZeneca. *'Device A3' Patient Information*. Jan. 2023. URL: https://den8dhaj6zs0e.cloudfront.net/50fd68b9-106b-4550-b5d0-12b045f8b184/fe598cda-d255-4446-998e-617607f61552/fe598cda-d255-4446-998e-617607f61552_pi_med_guide_rendition__c.pdf (visited on 09/01/2023).

[56] AstraZeneca. *'Device A1' (glycopyrrolate 9 mcg/ formoterol fumarate 4.8 mcg)*. URL: https://www.bevespi.com/ (visited on 09/01/2023).

[57] AstraZeneca. *Using Inhalers for COPD | 'Device A2'*. Oct. 2023. URL: https://www.breztri.com/administration/using-your-inhaler.html (visited on 01/21/2024).

[58] AstraZeneca. *About 'Device A3' (albuterol 90 mcg/budesonide 80 mcg) Inhalation Aerosol*. URL: https://www.airsupra.com/about-airsupra/about-airsupra (visited on 09/01/2023).

[59] AstraZeneca. *Instructions for Use 'Device A2'*. Jan. 2022. URL: https://www.breztri.com/content/dam/open-digital/breztri-consumer/en/pdf/instructions.pdf (visited on 02/21/2024).