

Master's Programme in Automation and Electrical Engineering

Real-Time Traceability and Monitoring System for Medical Device Batch Produc- tion

Otto Torkkeli

Copyright ©2026 Otto Torkkeli

Author Otto Torkkeli

Title of thesis Real-Time Traceability and Monitoring System for Medical Device Batch Production

Programme Automation and Electrical Engineering

Major Control, Robotics and Autonomous Systems

Thesis supervisor Udayanto Atmojo, PhD

Thesis advisor(s) Taru Rummukainen, MSc

Collaborative partner GE HealthCare Finland

Date 12.02.2026 **Number of pages** 85 **Language** English

Abstract

When designing a batch manufacturing process for a medical device critical to patient safety, it is of utmost importance to implement a traceable manufacturing system. The system must be capable of quickly and effectively identifying defects and anomalies in production, without disrupting the flow of batch manufacturing, while supporting fast root cause analysis.

This thesis presents a design and evaluation of a traceable production system for medical device batch manufacturing at GE HealthCare Finland. The work begins by examining the Process Failure Mode and Effect Analysis (PFMEA) and key design constraints, such as user needs, environmental conditions, manufacturing requirements, and equipment. These will act as the design inputs of the system and will be further developed into functional and non-functional requirements to a structured Software Requirements Specification (SRS) supported by Requirements Traceability Matrix (RTM).

The traceability system framework incorporated IoT-based data collection, serialization, cloud-based electronic Device History Record (eDHR) integration, while emphasizing the Digital Product Passport (DPP) fundamentals. The system development and validation were carried out iteratively through controlled manufacturing test builds, prioritizing functionality and high-risk requirements.

The system framework was able to mitigate high-risk failure modes from the affected production phases, as demonstrated using Pareto charts. The software requirements specification reviews provided a feasible, unambiguous, and verifiable system. System performance was evaluated against manual collection by comparing the data entry times, data collection efficiency, as well as workload reduction in the actual manufacturing environment. The automated framework was capable of reducing the data collection and processing times per unit. In addition, the operator workload was minimized to a single system interaction.

Keywords Traceability, Medical Device Manufacturing, PFMEA, SRS, DPP, IoT, Requirements Engineering

Tekijä Otto Torkkeli

Työn nimi Reaaliaikainen jäljitettävyy- ja valvontajärjestelmä lääkinnällisten laitteiden erävalmistukseen

Koulutusohjelma Automaatio ja Sähkötekniikka

Pääaine Sääntötekniikka, robotiikka ja autonomiset järjestelmät

Vastuuolettaja/valvoja Udayanto Atmojo, PhD

Työn ohjaaja(t) Taru Rummukainen, MSc

Yhteistyötaho GE HealthCare Suomi

Päivämäärä 12.02.2026 **Sivumäärä** 85 **Kieli** Englanti

Tiivistelmä

Potilasturvallisuuden kannalta kriittisten lääkinnällisten laitteiden valmistusprosessissa on tärkeää suunnitella ja toteuttaa jäljitettävä valmistusjärjestelmä. Järjestelmän on kyettävä tunnistamaan tuotannossa ilmenneet viat ja poikkeamat nopeasti ja tehokkaasti, sekä tukemaan nopeaa juurisyyanalyysiä loppuasiakkaalle asti.

Tämän lopputyön tarkoituksena on suunnitella, toteuttaa ja arvioida GE HealthCare Suomen lääkinnällisen laitteen tuotannon jäljitettävyysohjelmia. Järjestelmän suunnittelussa käytettiin riskipohjaista vikaantumis- ja vaikutusanalyysiä (Process Failure Mode and Effects Analysis, PFMEA) sekä keskeisten suunnittelurajoitteiden analysointia, kuten käyttäjätarpeiden, ympäristöolosuhteiden, valmistusvaatimusten ja laitteiston teknistä tarkastelua. Nämä muodostivat pohjan kehitettävän ohjelmiston vaatimusmäärittelyn (Software Requirements Specification, SRS) ei-toiminnallisten ja toiminnallisten vaatimusten osalta. Vaatimusten jäljitettävyysohjelmien (Requirements Traceability Matrix, RTM) luotiin kehitystyön alkuvaiheessa tukemaan riskipohjaisten ohjelmistovaatimusten priorisointia ja dokumenttien välistä jäljitettävyyttä ja sitä päivitettiin muutosten yhteydessä.

Jäljitettävyysohjelmassa hyödynnettiin IoT-pohjaista tiedonkeruuta, sarjanumerointia, pilvipohjaista laitehistoriatiedostoa (Electronic Device History Record, eDHR) sekä digitaalisen tuotepassin (Digital Product Passport, DPP) peruseräotteita. Järjestelmän kehitys ja validointi toteutettiin iteratiivisesti osana kontrolloituja testieräitä, joissa painotettiin korkean riskin vaatimuksia, sekä toiminnallisuutta.

Työssä kehitetty järjestelmä onnistui lieventämään tuotannovaiheissa tunnistetun korkean riskin vikaantumismuodot, jotka esitettiin Pareto-kuvaajien avulla. Ohjelmistomäärittelyn katselmoinnit varmistivat järjestelmän olevan yksiselitteinen, toteuttavissa, sekä todennettavissa. Automaattisen järjestelmän suorituskykyä arvioitiin vertaamalla automaattista tiedonkeruuta manuaaliseen tiedonkeruuseen tuotantoympäristössä. Automaattinen ratkaisu vähensi tiedonkeruu- ja käsittelyvaihtoja yksikköä kohden sekä pienensi operaattorin työkuormaa merkittävästi.

Avainsanat Jäljitettävyysohjelma, Lääkinnällisten Laitteiden Valmistus, PFMEA, SRS, DPP, IoT, Vaatimusten Määrittely

Table of contents

Preface and acknowledgements	7
Abbreviations.....	8
1 Introduction.....	10
2 Literature review.....	12
2.1 Traceability in Regulated Product Development.....	12
2.1.1 Definition of Traceability	12
2.1.2 Types and Levels of Traceability	12
2.1.3 Requirements Traceability Techniques.....	14
2.1.4 Importance in Medical Device Manufacturing.....	15
2.1.5 Traceability in Medical Device Regulatory Standards.....	15
2.1.6 Approaches for Enabling Traceability.....	17
2.1.6.1 Blockchain Traceability	17
2.1.6.2 Digital Product Passports	19
2.1.6.3 IoT-Enabled Traceability	20
2.1.6.4 Serialization and Unique Device Identification	24
2.1.6.5 Manufacturing Execution Systems (MES)	26
2.2 Risk Management in Safety-Critical Manufacturing.....	26
2.2.1 Overview of Risk Management in Safety-Critical Manufacturing	26
2.2.2 Risk Analysis Approaches.....	28
2.2.3 PFMEA in Manufacturing Processes	30
2.3 Software Requirements Specification (SRS) in Medical Device Manufacturing.....	32
2.3.1 SRS Overview in the Safety-Critical Domain	32
2.3.2 SRS Development Process	32
2.3.2.1 Requirements Elicitation.....	33
2.3.2.2 Requirements Analysis	34
2.3.2.3 Types of Software Requirements Specifications.....	35
2.3.2.4 Software Requirements Validation	36
2.4 Lifecycle Models in Safety-Critical Environments	39
3 Research Material and Methods	44
3.1 Traceability Assessment Methods.....	44

3.2	Risk Management Methods and Materials.....	46
3.3	SRS Development and Evaluation Methods.....	47
3.4	Documentation and Visualization Methods and Tools.....	49
4	System and the Software Requirements Specification Development Process.....	51
4.1	Development Process Overview.....	51
4.2	System Development Process	52
4.3	SRS Development Process	59
4.3.1	Acquiring Design Inputs from PFMEA	60
4.3.2	Acquiring Design Inputs from Design Constraints	62
4.3.3	Development of Functional and Non-functional requirements from Design Inputs	64
4.3.4	SRS Content and Structure.....	65
5	Results	68
5.1	Risk Mitigation Results.....	68
5.1.1	Risk Mitigation Results from the Initial Software Version	68
5.1.2	Risk Mitigation Results from The Revised Software	69
5.2	SRS Document and Requirements Evaluation.....	72
5.3	eDHR Records, Traceability, and System Performance Evaluation	74
5.3.1	Integrity and Traceability of the eDHR Records Audit Trail .	74
5.3.2	System Performance Evaluation.....	76
6	Conclusions and Future Work	78
	References.....	80

Preface and acknowledgements

I want to thank Udayanto Atmojo for supervising the thesis, GE HealthCare Finland for the opportunity to do my thesis as a part of their project, and the colleagues with whom I worked during the project for their support.

Helsinki, 12 February
Otto Torkkeli

Abbreviations

AIDC	Automatic Identification and Data Capture
CCP	Critical Control Point
CFMEA	Concept Failure Mode and Effects Analysis
CMT	Cause and Mitigation Table
DFMEA	Design Failure Mode and Effects Analysis
DHR	Device History Record
DPP	Digital Product Passport
eDHR	Electronic Device History Record
EEE	Electrical and Electronic Equipment
ETSI	European Telecommunications Standards Institute
EUDAMED	European Database on Medical Devices
FD&C Act	Federal Food, Drug and Cosmetic Act
FDA	US Food and Drug Administration
FMEA	Failure Mode and Effects Analysis
FTA	Fault Tree Analysis
GPRS	General Packet Radio Service
GPS	Global Positioning System
GTIN	Global Trade Item Number
GUDID	Global Unique Device Identification Database
HACCP	Hazard Analysis and Critical Control Points
HIBCC	Health Industry Business Communications Council
HRI	Human-Readable Interpretation
ICCBBA	International Council for Commonality in Blood Banking Automation
IEEE	Institute of Electrical and Electronics Engineers
IIoT	Industrial Internet of Things
IoT	Internet of Things
IOTA	Internet of Things Application
LAN	Local Area Network
LTE	Long Term Evolution
MES	Manufacturing Execution System
mIDoT	Micro Identifier Dot on Things
OEE	Overall Equipment Effectiveness
OPC UA	Open Platform Communications Unified Architecture
PBFT	Practical Byzantine Fault Tolerance
PFMEA	Process Failure Mode and Effects Analysis
PHA	Preliminary Hazard Analysis
PLC	Programmable Logic Controller
PMS	Production Monitoring System
Post-RS	Post-Requirements Specification
Pre-RS	Pre-Requirements Specification
QMS	Quality Management System
QR Code	Quick Response Code
RFID	Radio Frequency Identification
RoHS	Restriction of Hazardous Substances
RPN	Risk Priority Number
RTM	Requirements Traceability Matrix

SDL	Specification and Description Language
SLCM	Software Life-Cycle Model
SOA	Service-Oriented Architecture
SRS	Software Requirements Specification
STAMP	Systems-Theoretic Accident Modeling Process
STPA	Systems-Theoretic Process Analysis
SysML	Systems Modeling Language
TCP/IP	Transmission Control Protocol / Internet Protocol
TEEP	Total Effective Equipment Performance
UDI	Unique Device Identification
UDI-DI	Unique Device Identification Device Identifier
UDI-PI	Unique Device Identification Production Identifier
UML	Unified Modeling Language
WSN	Wireless Sensor Network

1 Introduction

Medical devices must comply with various country-specific regulatory requirements to protect the health and safety of patients. To ensure regulatory compliance along with quality, medical device manufacturing processes need to be properly documented, planned, as well as able to demonstrate sufficient traceability. Although regulations and traceability help to improve the medical device quality and reliability, together with ensuring that customer requirements are met, they often increase the overhead costs [1]. Real-time production monitoring systems (PMSs) enable automated data collection, including decision-making without human intervention, to reduce disturbances in production. These systems provide alarms in addition to proactively helping to detect issues in the production data [2]. While various PMS and traceability systems are available, as well as their effectiveness in improving traceability and efficiency has been researched in different industries, often special safety-critical manufacturing processes with regulatory requirements and specific equipment configurations make the adoption of commercial systems difficult.

At GE HealthCare, the current screen printing, oven curing, and thickness measurement production phase lacks an integrated, as well as an automated monitoring system, to meet the acceptable traceability and risk management requirements. While manual data collection options are available, they fail to address risk-based traceability and lack support for fast failure analysis, including the completeness of audit trails. Additionally, the manual collection significantly slows down the process due to additional labour required for data collection. As a result, a need for a structured monitoring framework for the medical device manufacturing process is evident.

This thesis proposes a traceable and effective monitoring system for screen printing, oven curing, and thickness measurement production phases at GE HealthCare Finland, which enables fast feedback, failure analysis, as well as compliance with the existing guidelines. The proposed system utilizes Open Platform Communications Unified Architecture (OPC UA), Modbus Transmission Control Protocol / Internet Protocol (TCP/IP), and Internet of Things (IoT)- enabled data capture combined with cloud-based electronic Device History Record (eDHR) integration. This system is expected to enhance product quality, traceability, and failure analysis while mitigating the risks and improving the efficiency of manual data collection.

This thesis aims to produce a complete and unambiguous Software Requirements Specification (SRS) documentation for such a system, enabling seamless and conflict-free software development phase, while ensuring the generation of a comprehensive electronic Device History Records (eDHR) for the manufactured product. The research methods of this thesis refer to the traceability frameworks literature review, risk-based requirement derivation, design input analysis from constraints and standards, development of the

SRS, and validation through established OPC UA, TCP/IP, and IoT-enabled data capture, with cloud-based eDHR integration. Finally, the proposed SRS is evaluated for completeness, correctness, and consistency. The performance of the proposed system is evaluated by comparing the automated traceability workflow to the manual data collection process. The eDHR events generated by the system are evaluated for the completeness and integrity of the audit trail.

The scope of this thesis will be limited to requirements engineering, risk management, process traceability design, SRS, eDHR model and evaluation of the SRS in the screen printing, oven curing, and layer thickness measurement production phase. Although the proposed eDHR model covers the entire production, it excludes the automated data capture and design choices outside the defined process. The programming phase and the final delivery of the system depend on the programmer. This thesis is part of a design transfer project at GE HealthCare, Finland, where a medical device is transferred for mass production. The author of this thesis is responsible for the design process of the monitoring system until a satisfactory system is achieved and approved. The author is also responsible for submitting the appropriate documents and deliverables to the company according to the documentation practices in use.

The sections of this thesis are structured as follows. Chapter 2 provides a literature review of the traceability approaches, riskmanagement, requirements engineering, and development process of a traceable production system for medical device manufacturing. Chapter 3 covers the research methodology used in the development process. Chapter 4 presents the development process of the system framework, creation of the Software Requirements Specification (SRS), and the elicitation of the design inputs based on the design constraints, risk management process (PFMEA), and traceability approach. Chapter 5 evaluates the proposed traceability system performance, risk mitigation, as well as SRS for completeness, testability, verifiability, and feasibility. The proposed eDHR collection model is evaluated for completeness of records and integrity of the audit trail. Chapter 6 provides a conclusion and discusses the potential extensions and future work.

This thesis addresses research questions on:

- How to design a real-time traceability system to support medical device manufacturing?
- How serialization, IoT-enabled traceability, cloud-based eDHR, and DPPs can be combined to automate traceability in manufacturing?
- How efficiently the proposed traceability system improves data collection efficiency, reduces workload, and supports end-to-end traceability?

2 Literature review

2.1 Traceability in Regulated Product Development

2.1.1 Definition of Traceability

Traceability is a widely used concept that refers to the ability to access information of a work item and identify it across its life cycle [3]. Also, ISO 9000:2015 standard, which covers the fundamentals and vocabulary of Quality Management Systems (QMS), defines traceability as the “*ability to trace the history, application of and object*”. Product or service traceability is associated with the processing history, the origin of parts and materials, and the final placement and accessibility of the product after delivery [4].

Important terminology regarding the concept of traceability includes trace (Noun), trace (Verb), trace artifact, and trace link. Trace (Noun) generally refers to the atomic nature of a trace or a chained connection between the traces. When using trace as a verb, it is defined as the action of following a trace link from a source artifact. Trace link consists of two trace artifacts, which contain an association from the source artifact to the target artifact. Trace artifacts are defined as trackable information units. In more detail, trace artifacts can be described as the marks or residual data in the systems and software development processes that are open to monitoring. Instances of different artifact types are design, requirements, and test cases [5, 6].

Traceability is often used together with requirements traceability, software traceability, and systems traceability. The distinguishing factor between these is the types of artifacts relevant to tracing. In the case of requirements traceability, it tries to find and connect requirements artifacts with relevant traces [5]. Software traceability focuses on establishing traces between software engineering artifact to other artifacts. Similarly, systems traceability concerns systems engineering artifacts to a variety of system-level components [6].

In this paper, traceability is referred to as the characteristic of a product requirement that can be monitored and documented from its original initialisation. In the context of medical device development, it means the information modelling across the device life cycle from requirements, design, manufacturing, and testing. It also covers the post-market monitoring of the product [1]. Further, traceability can be seen as a risk management and product life cycle management tool [3].

2.1.2 Types and Levels of Traceability

There are 4 common types of traceability, which refer to the nature of tracing in each type. Forward traceability enables the forward engineering process

by allowing a single requirement to be linked to the class method that implements it. The reverse engineering approach is obtained using backwards traceability to connect the implemented class method to the initial requirement. In forward tracing, development follows subsequent steps, which may not be in chronological order, such as from requirements through design to code in a systems and software development process. The process in backwards tracing is structured from previous steps, instead of following backward from code through design to requirements [5, 6]. These two types of traceability form the fundamental types of tracing. Further, bidirectional tracing comprises both forward tracing and backwards tracing [6]. Horizontal and Vertical tracing combine both backward tracing and forward tracing. While Horizontal tracing traces artifacts at the same level of abstraction, Vertical tracing enables end-to-end traceability by tracing artifacts at multiple levels of abstraction [5, 6].

To enable traceability view in the software development life cycle, two additional types of traceability were introduced, which can combine the previously discussed types. Pre-requirements specification traceability refers to the trace links that describe how requirements are derived from their original sources, such as stakeholder needs. Post-requirements specification traceability establishes trace links from the initial requirements to their implementation throughout the system development process. Figure 1 illustrates the concepts of Pre-RS traceability, showing links between stakeholders to the requirements specification artifact (S0) and Post-RS traceability, trace links to the requirements specification artifact (S0) from system development process artifact (S1) to (Sn) [6].

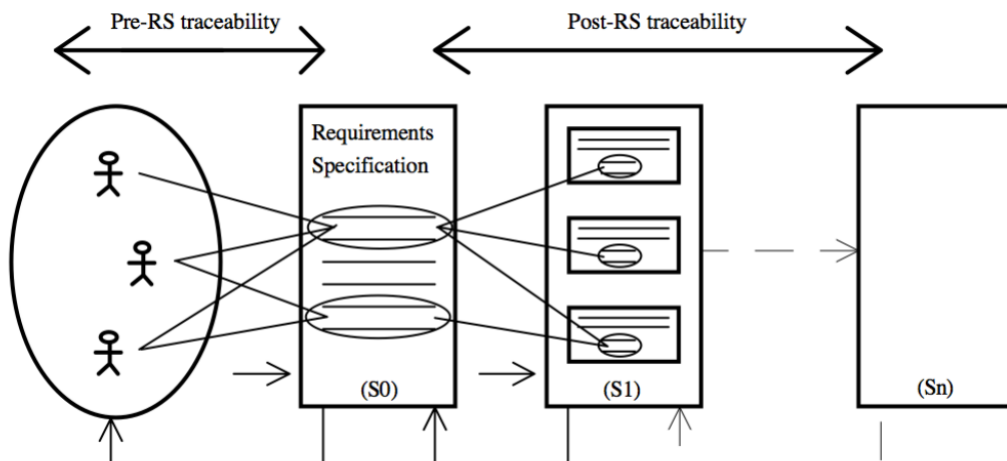


Figure 1: Pre- and Post-Requirements Specification Traceability [6]

Traceability can be achieved at different levels of comprehensiveness. However, the definitions for the levels tend to vary between industries and standards [3]. In this paper, we have previously mentioned end-to-end

traceability, which is an informal level of traceability referring to process traceability and requirements traceability of the entire development life cycle. Medical device standards are further discussed in Chapter 2.1.5 [6]. Traceability requires different levels of active involvement from systems or people. Automated tracing refers to the use of automated techniques to achieve traceability by maintaining trace links and creating artifacts. On the other hand, in manual tracing, traceability is established by a human tracer. A combination of human tracer involvement and automated techniques results in a semi-automated tracing, such as an individual is required to verify candidate trace links from automated techniques [5].

2.1.3 Requirements Traceability Techniques

Various techniques and tools exist to help visualize the traceability of the requirements, both in backward and forward tracing during the software development process. These techniques can help stakeholders, developers, and analysts to understand and analyse the life of the individual requirement development process using graphical notations [7].

Requirement Traceability Matrix (RTM) is a method used to express requirements in a matrix form, where typically columns of the matrix indicate the requirements, while the rows represent the relationships or artefacts. RTM helps to capture and link the requirements with related artifacts throughout the software development process. Figure 2 shows an example implementation of the RTM structure [7]. A traceability matrix is an effective tool for verifying that software requirements are linked to the corresponding test cases and ensuring test case coverage for each requirement. Information included in RTM, in addition to Figure 2, can include description, specification, type, and integration [8].

Requirement Traceability Matrix											
Test Case ID	TC_1	TC_2	TC_3	TC_4	TC_5	TC_6	TC_7	TC_8	TC_9	TC_10	# Test Cases for respective Requirement
Req. ID											
Req_1	✘		✘			✘					3
Req_2		✘			✘						2
Req_3			✘								1
Req_4				✘		✘					2
Req_5					✘		✘				2
Req_6						✘					1
Req_7					✘		✘				2
Req_8								✘			1
Req_9									✘		1
Req_10										✘	1

Figure 2: Example Chart of Requirements Traceability Matrix [7]

Hierarchical traceability refers to a graphical structure of traceability that is represented using requirements as leaf nodes, and titles and other information are visualized using internal nodes. Links and structure of the hierarchical visualisation can be easily followed, and complete information for requirements traceability obtained if the dataset and number of traceability links are not significantly large [7, 9].

Other relevant requirements traceability methods are cross-referencing and list traceability. The goal of cross-referencing is to systematically list the artefacts relevant to tracing and establish relationships lists to other artefacts containing relations. The method is easy to understand for a user, but the structure is prone to problems relating to displaying a single link in the table. List traceability contains information about the origin, destination, and relationships of the requirements. Its benefit is that it is widely applicable, but it supports only a one-way entry method [7].

2.1.4 Importance in Medical Device Manufacturing

Medical devices are a critical part of modern healthcare systems. Their significance in supporting human health has led to strict controls in marketing, manufacturing, and use on a national and worldwide scale. [10] As a result, safety-critical domains, such as the medical device domain, must ensure secure, safe, and reliable systems. Traceability plays a key role in achieving such systems [11].

Utilizing risk management practices to implement traceability to medical device development helps to identify risk mitigation activities and hazards [11]. Traceability-driven systems improve the medical device quality by determining whether customer requirements, government-set standards, and regulatory requirements for the medical device are complied. In addition, traceability allows designers to efficiently track the development of design solutions from the original requirements [1].

Traceability can improve the efficiency of manufacturing and enable fast responses in case of recalls. With the implementation of traceability, the location of the product within its lifecycle, from raw material to complete product can be accurately identified. Finally, performance metrics such as Overall Equipment Effectiveness (OEE) and Total Effective Equipment Performance (TEEP) can be used to evaluate how effectively machines are being utilized [3].

2.1.5 Traceability in Medical Device Regulatory Standards

This subchapter expands on the medical device and standards, and guidelines. Further, a description of the most commonly used standards and guidelines for traceability assessment in medical devices.

ISO 13485: Medical Devices – Quality Management Systems (QMS), Requirements for Regulatory Purposes is an international standard that defines criteria for best QMS in all phases of the medical device lifecycle, such as design, development, and production. General requirements include documentation of QMS effectiveness and the role taken by the organization. Also, implementing process controls based on a risk-based approach and validating requirements of the computer application are considered general requirements for effective QMS. The company will obtain ISO 13485 registration once the QMS with the required criteria is established, and the registrar audit is satisfied. Traceability is more specifically mentioned as a part of product realization planning to identify activities required to ensure product traceability. Production and provision for service traceability should cover records of materials, components, and work environment conditions used [12].

ISO 14971 - Application of risk management to medical devices standard is utilized to ensure medical device risk management requirements compliance. The standard is used by the manufacturer to define the process of estimating and evaluating associated risks, implementing risk control, and effectively monitoring the risk management controls. General requirements for the manufacturer must establish, implement, and document the processes of hazard identification, risk estimation and evaluation, risk control, and effective monitoring of the risk controls. Elements of Risk analysis, Risk evaluation, Risk control, and production and post-production process activities shall be provided to cover the entire life cycle of the medical device. The risk analysis, evaluation, implementation, and verification of risk controls and results must be traceable for every hazard identified. Further guidance on this is provided in the ISO/TR 24971 Medical devices – Guidance on the application of ISO 14971 [12].

The US Food and Drug Administration (FDA), with its legal authority (FD&C Act) Federal Food, Drug and Cosmetic Act is responsible for developing, publishing and implementing regulations for medical devices. FDA-CFR Title 21-Food and Drugs: Part 820 Quality System regulation states that traceability and identification processes shall be maintained and established by the manufacturer to cover installation, production, distribution, and receipt for medical devices, including the corrective actions [12].

EU 2015/863: Restriction of Hazardous Substances (RoHS) -3 sets restrictions to the use of hazardous substances in electrical and electronic equipment (EEE) in order to protect human health, the environment and free competition and trade. Manufacturers obligation is to provide technical documentation and create, update, and maintain requirements set out by the legislation. An EU Declaration of Conformity and affix the CE mark to the completed product is required from the manufacturer. EEE should be clearly described to enable identification and traceability. EU 2017/746 – In Vitro Diagnostic Medical Devices regulation sets the rules for placing in vitro diagnostic medical devices available and on the market within the European

Union. Article 23 states that manufacturers need to achieve an adequate level of traceability while cooperating with distributors and importers [12].

2.1.6 Approaches for Enabling Traceability

While regulatory frameworks and requirements traceability are an essential part of ensuring compliance and documentation, they do not specify any technical solutions for how traceability should be achieved in practice. A variety of different approaches have emerged from academic research and industry to enhance traceability in the product lifecycle and manufacturing. Instead of manual approaches, these techniques aim to automate the traceability and Device History Record (DHR) collection throughout the product lifecycle, improving the efficiency and accuracy of the traceability.

2.1.6.1 Blockchain Traceability

Blockchain is a distributed ledger, which is formed by a chronological chain of blocks containing valid internet activities since the addition of the last block. Each block in the chain includes encrypted pieces of information visible to anyone at any time, however, modifying the data in the blockchain requires authorization. As a result, blockchain forms a complete and unchangeable record of network activities, which can be accessed by everyone in the network. Blockchain enables different entities to exchange values in the blockchain securely, without knowing each other, through a decentralized network, eliminating the need for third-party intermediation. In addition, implementing improvements to a system can be introduced by anyone, if all parties have accepted the change. This enhances the transparency and legibility [13].

Recently, various studies and research have been conducted to implement blockchain technologies into the manufacturing domain and as well as medical device supply chains. Blockchain fulfils various healthcare management supply chain requirements, such as product identification, tracing, verification, detection, and notification [14]. Different frameworks have been researched and presented, especially in the pharma industry, to detect counterfeit medicines, enhance drug traceability and authenticity, as well as traceability of drugs used by a pharmaceutical company. As a result of the decentralized structure, medical device manufacturers can understand the drug requirements better, enhancing transparency. In addition to the manufacturer, the customer can also verify the authenticity of the drug [15].

Xiaoling Xia et al. proposed a blockchain-driven medical device traceability system, where the architecture design is divided into data layer, network layer, consensus layer, incentive layer, contract layer, in addition to the application layer. Data layer stores the device information using Merkle tree structures with timestamped blocks, while the network layer ensures data

dissemination and verification. The consensus layer adopts the Practical Byzantine Fault Tolerant (PBFT) protocol to maintain consistency among alliance-chain nodes without the computational overhead. Smart contracts in the contract layer govern the registration, update, and transfer of medical device records, enabling automated enforcement of traceability rules. The application layer then provides interfaces for manufacturers, distributors, hospitals, and consumers, allowing each stakeholder to input or query lifecycle data according to their permissions. Figure 3 visualizes the functionality of the proposed framework, where manufacturers upload product attributes to generate unique blockchain-based device IDs, distributors and hospitals append verified transaction records as devices move through the supply chain, and consumers can retrieve complete provenance information from RFID without needing authentication. This integrated design enhances transparency, integrity, and end-to-end traceability across the entire medical device lifecycle [16].

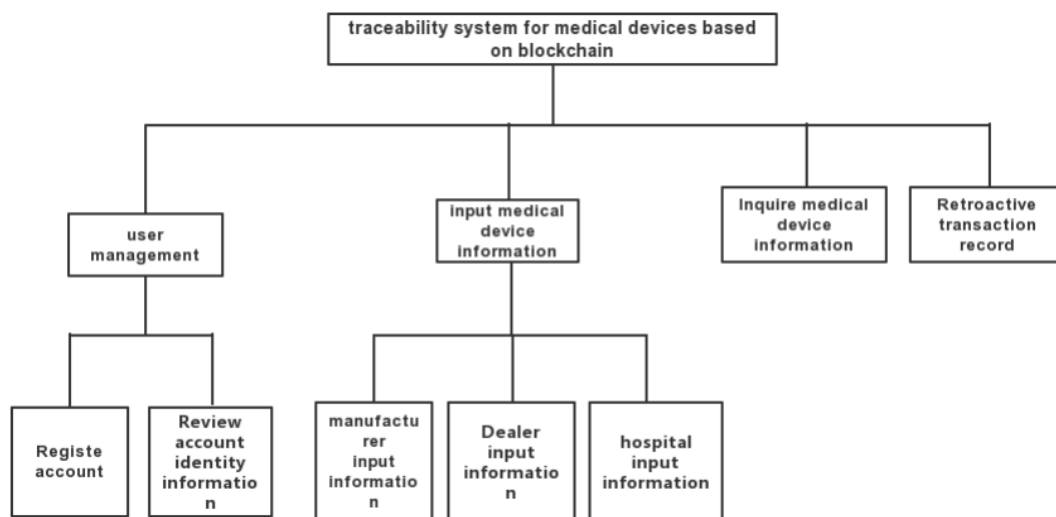


Figure 3: System Function Module Proposed by Xiaoling Xia et al. [16].

Iyolita Islam and Muhammad Nazrul Islam proposed a framework for safe and secure medicine production and distribution to mitigate the issue of counterfeit drugs using blockchain. This framework defines different entities, such as manufacturers, distributors, retailers, and consumers, as authenticated nodes. These authenticated nodes were integrated into a blockchain network built using Hyperledger Fabric. Further, each medicine unit is given a unique digital identifier, such as a QR code containing manufacturing information, batch number, manufacturing date, packaging information, as well as expiration date, which are registered to the blockchain as immutable records. When medicines progress in the supply chain, each transfer between the nodes triggers the creation of a new block that records timestamps in addition to chronological traces of ownership, location, and transaction details.

Similarly, as proposed by Xiaoling Xia et al., smart contracts are utilized to automate the validation as well as rule enforcement by verifying transaction inputs and identifying inconsistencies such as duplicated codes. If anomalies arise, the system can alert manufacturers and regulatory authorities. The framework was able to ensure transparency for drug traceability, and the prototype showed potential for practical scalability for a reliable solution for preventing medicine counterfeiting. However, limitations of this study included dependency on accurate data, QR codes vulnerability to cloning, and high implementation costs, as well as a limited simulation environment [14].

2.1.6.2 Digital Product Passports

Recently, the European Commission has suggested the use of digital product passports (DPPs) in the European Single Market as a part of its Green Deal. The Digital Product Passport (DPP) refers to a concept in which a digital representation of a product's information throughout its lifecycle is created. DPP provides a repository of data containing information, such as origins, materials decomposition, usage guidelines, as well as environmental impacts. As a result, DPP is considered to enhance product traceability, promote sustainability, and facilitate the circular economy. Currently, the DPP covers a variety of different possible use cases and understandings. However, many of these are conceptual, and there are gaps in the current research, as well as a lack of standardized frameworks [17].

Recent research by Hendro Wicaksonoa et al. discusses the recent technological advancements to support DPPs, as well as proposes a conceptual DPP architecture and adoption framework, combining a systematic literature review from various relevant studies. As a result of the paper, three different tiers of DPPs were identified, including data gathering, data curation, processing, and sharing, and data use and exploitation. Digital product passports rely on a continuous data lifecycle that begins with the collection of product-related information supplied by manufacturers and captured throughout the product's lifespan, either automatically or at key lifecycle milestones. This data is gathered using IoT devices along with embedded sensors containing data such as unique identifiers, component and material specifications, chemical content, lifecycle performance indicators, environmental and social impact metrics, repair and recycling instructions, and ownership and compliance records, which are often incorporated into physical identifiers or QR codes. At the second tier, the data is then curated, processed, and shared using digital technologies to translate information from physical systems into actionable insights that enhance product design, support informed decision-making, and connect value chains. Enabling technologies identified contained distributed ledger technologies, blockchains, digital twins, decentralized identifiers, as well as artificial intelligence. As a result, a six-layer DPP architecture was proposed. The proposed architecture includes an IoT-

driven data collection layer, a data processing layer with gateways, Internet of Things Application (IOTA) network, and a smart contracts-driven distributed ledger layer, a digital passport layer generating complete passports, a data management and interoperability layer, along with an application layer for data analysis, stakeholder access, and the circular economy visualized in Figure 4 [18].

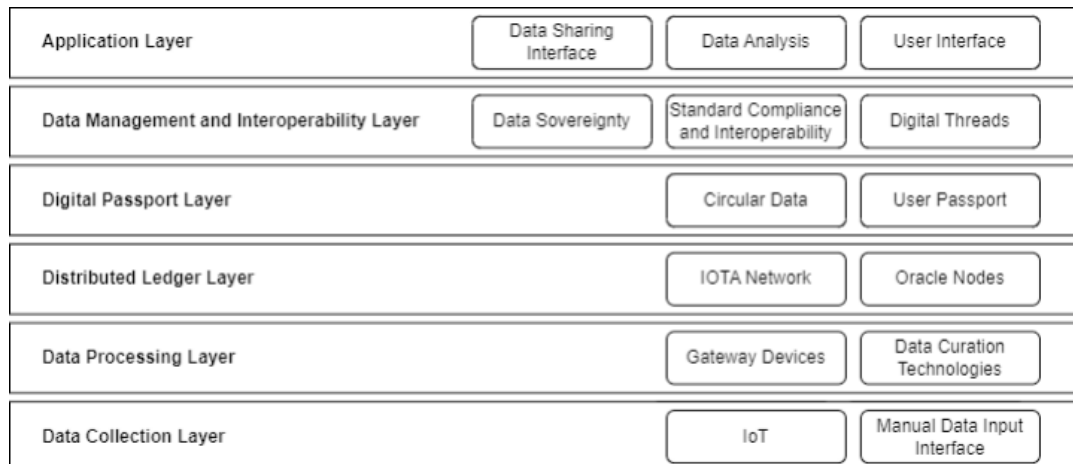


Figure 4: DPP Architecture Proposed by Hendro Wicaksonoa et al. [18]

Finally, at the third data use and exploitation tier, the processed data is used to provide optimized product design, transparency, improved lifecycle assessment, enhanced resource efficiency in manufacturing, circular economy models, as well as improve the end-of-life decisions. Despite the exciting development of DPPs, the current research contains limited insights into the design and implementation of scalable software platforms. Additionally, multi-actor perspectives require deeper knowledge of how data differs among supply chain actors in order to facilitate the successful implementation of DPPs [18].

2.1.6.3 IoT-Enabled Traceability

Internet of Things (IoT) refers to the idea of everyday objects being equipped with sensing and communication technologies, allowing them to be identified, located, monitored, and controlled over the Internet, regardless of the communication method used. It is essentially a combination of different software and hardware technologies, which allow data storage, retrieval, and processing in different electronic systems [19].

While various IoT frameworks have been proposed by parties, such as Cisco, the Institute of Electrical and Electronics Engineers (IEEE), and European Telecommunications Standards Institute (ETSI), there is a lack of a standardized framework. One of the most used frameworks includes dividing the IoT architecture into 3 layers, consisting of the Application Layer,

Transmission Layer, and Perception Layer, as seen in Table 1. These can be further divided into sub-layers, based on functionality as adopted from [20].

Layers	Sub-layers	Key Features	Key Technologies
Application Layer	IoT Applications	Handheld Devices, Terminals and User Interface	Cloud Computing, Middleware, M2M, Service Support Platform
	Application and Support Layer		
Transmission Layer	Local & Wide Area Network	Connectivity Establishment and Information Transmission	Internet, GPRS, Wi-Fi, Ad hoc Network
	Core Network		
	Access Network		
Perception Layer	Perception Network	Sensing, Identification, Actuation and Communication Technologies	RFID, WSN, GPS, Bluetooth
	Perception Nodes		
Network Management	Physical and Information Security Management		Trust Management

Table 1: IoT Architecture Layers [20]

The perception layer in the IoT architecture refers to the technologies responsible for sensing (collecting data from the environment to databases or Cloud), identification (assigning and using unique identities to detect and track objects), actuation (actuating devices based on the collected data), and communication (facilitating data exchange among heterogeneous devices) with minimum human involvement. This layer is further divided into Perception nodes and Perception network. The nodes are, for instance, physical devices, sensors, actuators, or controllers, which are able to form a mesh network or multi-hop environment to enhance scalability. Depending on the environment and technology, the common devices used as nodes include Radio Frequency Identification RFID readers, Quick Response (QR) code or Barcode readers, Global Positioning System (GPS) devices, Bluetooth devices, and different sensors, including light, humidity, and temperature sensors capable of gathering information from the environment, object identification, data control, and object control. The purpose of the Perception Network is to distribute the data collected by the nodes for further transmission using wired or wireless communication. After perception, the generated data is transmitted to the information processing units for analysis, data mining, data aggregation, and data encoding at the Transmission Layer. This layer also provides functionality for network management. At the sub-layer level, further division is made to access the network, core network, in addition to the local and wide area networks. The access network is a type of

telecommunication network providing a bridge between the subscriber and service providers. Communication technologies such as mobile communication, satellite communication, and wireless communications enable connectivity to the end users with technologies, for instance, Wi-Fi network, Zigbee, Low Energy Bluetooth, as well as 4g-LTE and 5G. The access network can be centralized or non-centralized depending on the system architecture. The core network, most often the Internet, provides the fundamental framework for IoT systems. Its purpose is to transmit data to end users who are connected through access networks and function as a backbone to exchange information as well as services. Local and wide area network functions as an interconnection between devices in a smaller region. Devices connected to the local area network (LAN) are able to communicate directly with themselves and with remote devices utilizing gateways. Recent research in Low Power Wide Area Networks has gained attention as they support connectivity with low-power devices [20].

At the top of the IoT architecture stack is the Application layer, which manages and provides applications based on the information obtained from the perception layer. It is visible to the end user and provides access to tailored services to the end user over the network through various handheld devices and terminal equipment. The application support sub-layer offers enterprise services by executing computations, processing, recognition, and filtration over data. This layer utilizes Services Oriented Architecture (SOA) to support Quality of Service (QoS), as well as other service management functions. To deliver its functionalities, the application support layer uses middleware to perform computation and hosts servers. Additionally, it includes cloud computing for large-scale data storage and processing, Machine-to-Machine (M2M) application models to enable connectivity between devices through wired or wireless links, and Service Support Platforms consisting of user interfaces for applications and auxiliary services. Recent advances have focused on the Cellular Wide Area M2M connectivity and Low Power Wide Area M2M technologies, such as LoRa and SigFox [20]. As for Industrial IoT (IIoT), often associated with combining Industry 4.0 and IoT, typical data acquisition protocols such as Modbus, Profibus, or OPC are integrated with the application layer [21]. Modbus protocol establishes bidirectional communication between Modbus slave devices and the sensed data. Programmable Logic Controllers (PLCs) can be configured as a master, while various field devices act as slaves, enabling applications to access and utilize data efficiently. Figure 5 shows an example implementation of Modbus enabled Arduino application [22].

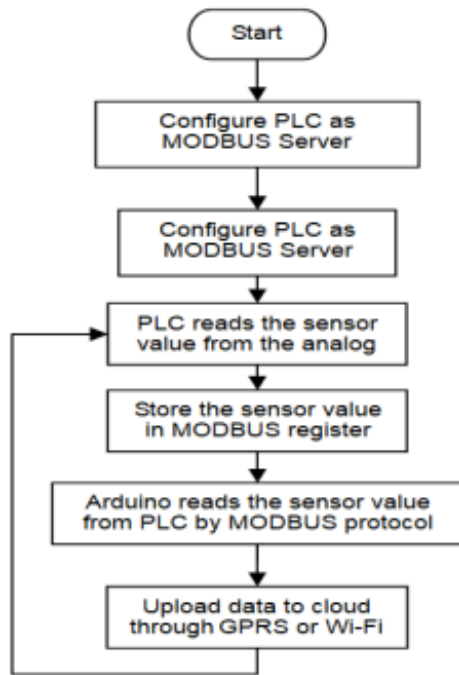


Figure 5: Modbus-Enabled M2M Communication Configuration [22]

OPC UA provides secure, seamless, and interoperable M2M for IIoT (Industrial Internet of Things) applications among heterogeneous devices and networks. It uses a client-server model, where data is represented using an object-oriented structure. Information elements are represented as nodes containing attributes, such as an identifier, name, and data type. OPC UA stores the data in nodes across different servers, which can be accessed by clients. This data can be accessed to perform real-time analysis and visualization across various industrial environments [23].

The IoT applications are divided into information collection, analytics, and real-time decision-making applications. Further, the information collection applications are responsible for collecting the data from perception nodes, as well as storage, analytics involves offline preprocessing of the collected data, creating a general model for evaluating future data, and real-time decision-making applications are responsible for executing appropriate actions based on the collected data [20].

Research has been able to demonstrate that the use of IoT increases traceability capabilities in Supply Chain Management. IoT devices can ensure authenticity, along with supporting tracking of the origin and quality of a product. In addition, these devices support real-time tracking, visibility, and traceability covering the entire supply chain, including manufacturing. The most common way of storing the large amount of data generated by the IoT devices is through cloud-based centralized infrastructures. As a result, privacy and security challenges may pose challenges for risk mitigation. The

previously covered blockchain technology has the possibility to enhance IoT security through decentralized structures, enhancing data integrity [24].

2.1.6.4 Serialization and Unique Device Identification

Effective product identification is critical to any traceability system, enabling each physical item to be uniquely recognized and linked to its associated data throughout its lifecycle. Modern manufacturing environments, and especially manufacturers producing miniaturized components, have faced challenges due to the decreasing surface areas and, on the other hand, increasing demand for precision in unit identification. Indirect marking technologies, such as RFID tags and labels, as well as direct part marking methods including laser marking, lithography, and micro-scale identifiers, are used to ensure reliable serialization and tracing [25].

Indirect part marking refers to the product identification that relies on a separate id carrier. Further, the separate carrier identification is typically a sticker containing a numeric, alphanumeric, bar, or 2D code. The indirect part making is usually used on larger parts. Additionally, RFID passive tags are used for part identification in large companies such as Airbus. RFID tags, combined with IoT, have been utilized in traceability systems to enable different manufacturing resources to communicate and interact with each other in real-time. RFID tags vary in size, allowing even small part identification using, for instance, ultra-small RFID chips that can be as small as 0.3mm x 0.3mm [3]. A barcode can be read using lasers, which identify the lines or squares marked on an item and retrieve a string to identify different characters, which will form the identification. The unique device identification has to be marked onto the surface of the medical device and follow the standards for GS1, Health Industry Business Communications Council (HIBCC), in addition to ICCBBA, especially in the US. GS1 barcodes contain (GTIN) Global Trade Identification, an expiration date (day, month, year), as well as a batch or lot number. The size difference between the 2D codes compared to the 1D barcode containing the same data is visualized in Figure 6, where each mark encodes the same data. Typically, many industries use linear barcodes more often than 2D barcodes. The larger sizes allow readability in two directions, in addition to the possibility of orienting the codes on a product's packaging. However, medical devices' 2D barcodes are preferred as they have a larger data storage capacity, smaller size, as well as support for omnidirectional readability. The most common 2D codes include the QR-code (Quick Response), data matrix, in addition to PDF417, which is bigger in size compared to the QR-code and data matrix [25].

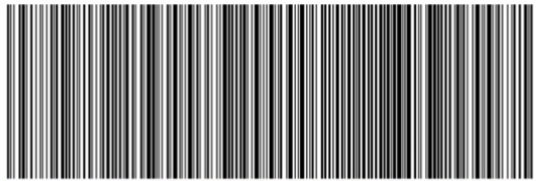


SICK AG, Nimburger Strasse 11, 79276 Reute	OCR/OCV
	1D Code
	2D Stacked Code
	2D Matrix Code

Figure 6: 1D and 2D Barcode Size Comparisons With Same Data [25]

Direct part marking refers to the method where the part's surface is marked to identify the part or product. The common methods for direct part marking include laser marking and lithography, in addition to the newer approach of Micro Identifier Dot on Things (mIDoT) [3]. Laser marking is practical when durable identification is required on small parts. Specifically, Data Matrix codes can be effectively read on compact surfaces. With the use of lithography, extremely small micro scale device to be identified precisely. This is achieved through engraving micro-QR or micro Data Matrix codes to the part in environments challenging to traditional marking methods, due to the resolution requirements. The codes can be engraved using microfluidic devices as well as drug capsules with microscopic QR codes. Typical sizes of the code can be 0.9 mm x 0.9 mm in microfluidic devices and 0.3 mm x 0.3 mm using projection lithography. This method is highly important in microelectronics, lab-on-chip, and anti-counterfeiting [25]. Finally, mIDoT approach, which is a newer method used to specifically identify small parts using glitter ink dots less than 1 mm in radius. Each dot forms a unique random pattern, after which the identification is done by matching the image to the unique glitter pattern. This approach benefits from low material costs and can be practical for parts too small for traditional approaches, but on the other hand tends to struggle with data storage in addition to processing time due to the image recognition process [3, 25].

As for medical device identification regulations set by the FDA and the EU Commission, a Unique Device Identification (UDI) shall be marked directly onto the device for reusable medical devices. However, this excludes the single disposable devices from requiring an official UDI marked directly onto the device. The UDI itself is a globally accepted device identification framework, which is created from alphanumeric or numeric characters. Both of these parties comprise the UDI device identifier from UDI-DI (Device Identification), containing identification for the manufacturer and device model acting as the database key, and UDI-PI (Production Identifier), consisting of

information such as LOT/batch number, manufacturing date, and serial number. They also require the UDI to appear both in (Automated Identification for Data Capture (AIDC) and human-readable interpretation (HRI) representation. The issuing agencies for the official UDIs are, for instance, the previously mentioned HIBCC, GS1, and ICCBBA. In the EU, the manufacturer must submit the relevant information to European Database on Medical Devices (EUDAMED) and in the United States to Global Unique Device Identification Database (GUDID) [26, 27].

2.1.6.5 Manufacturing Execution Systems (MES)

Manufacturing Execution Systems (MES) play a vital role in the modern manufacturing domain. They are increasingly more often identified to be foundational enablers of real-time visibility, traceability, as well as process control within Industry 4.0 systems. Due to the need for higher connectivity, flexibility, and data-driven decision-making, MES systems can provide an interface between operational technology (OT) and information technology (IT). Recently, studies have shown that MES not only function as a tools for production monitoring, but they have also become critical components of smart factories. They are able to synchronize real-time processes, support part- and material traceability, as well as provide fast responses to deviations or unexpected disturbances in the process [28].

Zwolinska et al. describe MES as an integral system, which is capable of monitoring, managing, and synchronizing real-time manufacturing operations, additionally supporting traceability and origin of the components throughout the production life cycle. They showcased that MES is able to function as a hub for information connecting machines, operators, and business systems. As a result, manufacturers can achieve responsive, real-time, and transparent in environments with high variability as well as strict quality requirements [28]. Mantravadi et al. discussed that MES lays out the foundations for IT and OT integration, which is essential for real-time data exchange and the development of Industry 4.0 systems. Their analysis demonstrated the importance of MES for distributed and legacy equipment, further improving transparency, as well as supporting the detectability of production issues and interoperability requirements of IIoT systems [29]. Also, previous review by Mantravadi and Møller discussed that the MES technologies are considered to be critical in obtaining real-time visibility, reducing downtime, improving quality, as well as enabling comprehensive analytics and decision making [30].

2.2 Risk Management in Safety-Critical Manufacturing

2.2.1 Overview of Risk Management in Safety-Critical Manufacturing

Manufacturing-related faults, such as functional problems, packaging, and software glitches, are one of the most common causes of recalls in medical devices, endangering patient safety. Risk Management is a critical part of the medical device development lifecycle, which offers a method to assist professionals in medical device development to ensure safety, reliability, and desired behaviour of the product. The ultimate goal is to minimize the possibility of a fault occurring in a product. There exist multiple guidelines, practices, and processes to systematically construct a risk management lifecycle for medical devices, with a common structure described in Figure 7 [31]. As previously covered in section 2.1.5, the ISO 14971 - Application of risk management to medical devices standard defines the requirements that the manufacturers must establish and demonstrate to ensure compliance with risk management for medical devices [12]. In order to comply with the requirements set by the ISO or FDA, an effective risk management framework and documentation are essential for the development of the medical device [31].



Figure 7: Risk Management Steps [31]

The risk analysis step describes the initial assessment of potential risks derived from the medical device's intended use. It is essential to be able to identify any potential risks as early as possible to accommodate the risk assessment process. The risk analysis should not only focus on identifying the risks and their causes but also address the hazards related to the risks. Risk evaluation addresses the level of likelihood and severity of the risks. This helps to assess risks based on how severe the risk is and how often it is likely to occur compared to other risks identified. Visualizing the risks based on these factors is an effective way to determine which risks should be prioritised. When risk has been identified and evaluated, appropriate risk controls may need to be established. The purpose of risk mitigation is to decrease the

overall risk to an acceptable level. In order to achieve the required level of severity, design changes and protective measures such as reducing the likelihood of the risk have to be made. Multiple risk mitigation strategies may need to be incorporated to eliminate and reduce the severity of the risks to the level satisfying the standards. The risk control phase ideally begins already in the design input phase, covering the entire lifecycle of the product.

Finally, documentation of the risk management plan and methods is a key to successful risk management. The documentation of the risk management should contain reports, evaluations, diagrams, and steps from the entire risk management process. Also, after a finished development project, it is important to report the effectiveness of the control actions as well as keep the documentation up to date, since the risk management strategy is a critical part of the entire life cycle of the product development process [31].

2.2.2 Risk Analysis Approaches

To access and manage risks, many standard operating procedures and tools exist to support the risk management process. The basic structure of risk management can include decision-making through Flowcharts, Check Sheets, Process Mapping, and Cause and Effect Diagrams [32]. The common structured methods for comprehensive risk management are covered in this chapter.

Preliminary Hazard Analysis (PHA) is a risk analysis tool that utilizes previous information, such as experience, to identify future hazards and hazardous situations and determine potential corrective actions. Probability of occurrence is estimated for a specific activity, facility, product, or system, and the risks are ranked using likelihood and severity evaluation. PHA can be divided into four main steps covering PHA prerequisites, Hazard identification, frequency and consequence estimation, follow-up actions, and risk ranking. PHA is effective, especially if used early in the product development to determine key inputs to safety. It helps to comply with standards such as the ISO 14971 standard, particularly when combined with FMEA (Failure Modes and Effects Analysis), to further implement sufficient controls and analyze device failures [32].

FMEA is a structured method widely used in the healthcare industry to proactively mitigate and identify process or product failure modes. Successful implementation of the FMEA method minimizes the effort required to design out failures from the system, utilizing previous experience with similar processes. Like PHA, effective FMEA covers the entire life cycle of the product. Ultimately, FMEA aims to produce actions that decrease the likelihood and severity of failures beginning from the highest-priority risks. As seen in Figure 8, FMEA can be divided into Concept FMEA (CFMEA), Design FMEA (DFMEA), and Process FMEA (PFMEA), which will be covered more thoroughly in chapters 2.2.3. CFMEA typically covers the system and subsystem

level early in the project initiation, focusing on the possible failure modes of the functions and concept proposals. DFMEA is used in the design phase to identify and prevent failure modes of the product. It is considered the most crucial type of FMEA as it is closely related to the validation of the design parameters in order to achieve the desired functional performance level at the system, sub-system, and component level [33].

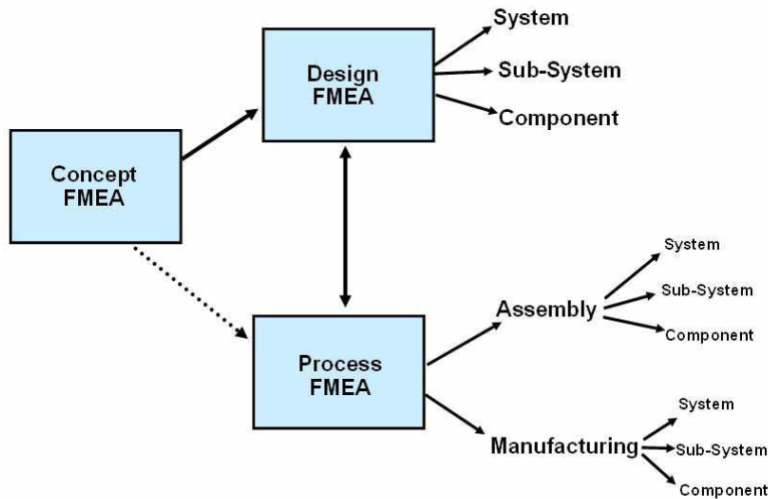


Figure 8: Different FMEA Types [33]

FTA (Fault Tree Analysis) is a top-down method that starts with a functional failure and builds on it to find the root causes of the failure event of a process. Failure events at various levels can be accessed from a visual representation, which is constructed using logical modelling and Boolean algebra, eventually forming a tree of the fault modes. FTA allows a detailed description of the relationships between the events impacting quality [32, 34].

HACCP (Hazard Analysis Critical Control Points) is a proactive risk management method used to ensure the reliability, quality, and safety of the medical device. The core of this approach includes identifying the Critical Control Points (CCPs) in a process and focusing on identifying potential hazards at each of these points. HACCP aims to study specific risks at the points and establish critical limits for these points. Further, the limits will determine the acceptable values for a control point. Control points are constantly monitored to ensure that the values are within acceptable limits. If there are problems or deviations, corrective actions can be taken quickly. In order to ensure traceability and demonstrate standards compliance, each phase is documented [35].

More recent approaches, such as Systems Theoretic Process Analysis (STPA), have emerged from the Systems-Theoretic Accident Modeling Process (STAMP), which views safety as a control problem instead of component failures. This approach models the systems as a control structure consisting of sensors, controllers, and feedback loops. The controller of the system can

be a human or software that executes certain actions that have an impact on the process. Feedback is provided to the controller by sensors, allowing the controller to adapt its actions based on the model. In the STPA approach, hazardous states can occur from various system factors, such as uncontrolled component failures and unsafe interactions among components. The advantage of STAMP models, such as the STPA model, is that it is demonstrated to reduce subjectivity, identify requirements flaws, and provide a more useful picture of the system, resulting in a more complete model. The STPA process consists of iterative steps of identifying potentially hazardous control actions and determining the safety of the identified control actions. However, the system boundaries and goals need to be defined in advance in order to elicit the system-level hazards. The initial step is to identify the inadequate safety controls. The control action can be hazardous by being unsafe, not provided or followed, safe control action is provided too late, too early, or out of sequence, and continuous safe control action is applied too long or stopped too soon. As a result of the analysis, STPA provides appropriate controls for design and operational recommendations. Further, it enables component failures as well as unsafe interactions to be controlled through design, process, or social controls [36].

2.2.3 PFMEA in Manufacturing Processes

This subchapter further examines the PFMEA risk management method in manufacturing. PFMEA is a structured risk management tool used widely in the safety-critical product manufacturing domain, where reliability and precision are essential, such as in medical device manufacturing, to analyse, detect, and prevent potential failure modes. It provides a tool for meeting international standards and improves the product quality, process reliability, and process efficiency by enabling corrective actions before potential production issues occur [37].

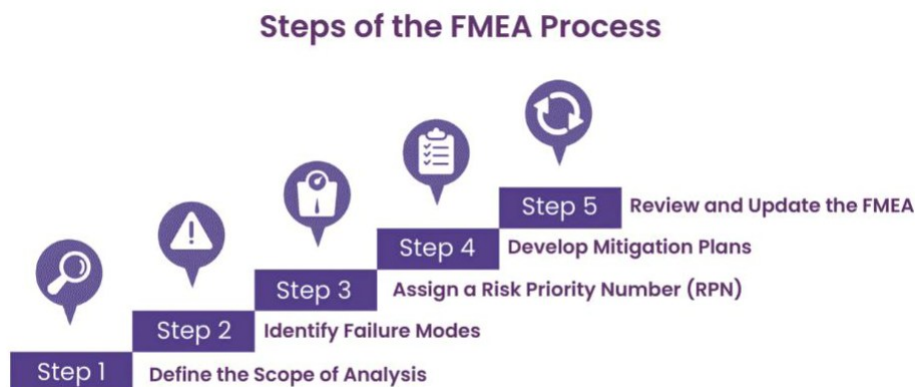


Figure 9: FMEA Process Steps [37]

failure modes can be interpreted as no function, where the system does not work at all, or over-function, referring to a failure mode where the system works, but performance gets gradually worse. Various techniques are used to identify failure modes effectively, including brainstorming, storytelling, cause-and-effect diagrams, and reviewing previous failure modes on similar risks. Cause and detection analysis highlighted yellow in Figure 10 objective is to study each failure mode individually and expose the causes of the failures. Failure mode can be associated with multiple causes. The detection phase in this section refers to the process of identifying current process controls and activities established to detect or prevent failures. The aim of the controls for preventive controls is to minimize the likelihood of failure and for detection controls to maximize the possibility of detecting a problem in advance before reaching the customer. The purpose of improvement actions is to introduce actions or evaluations from engineering that will lower the possibility or overall risk to acceptable levels. These are prioritised using the previously discussed RPN values [39, 40].

2.3 Software Requirements Specification (SRS) in Medical Device Manufacturing

2.3.1 SRS Overview in the Safety-Critical Domain

Requirements engineering involves the management, documentation, creation, and maintenance of a set of requirements for a product. The engineering perspective ensures that the Software Requirements are complete, consistent, valid, and verifiable [41]. A software requirement can be defined as what the software must be able to do in order to comply with a standard or technical specification, or what the user needs from the software to be able to achieve. It can also mean a documented representation of the previous definitions [42]. Software Requirements Specification (SRS) is a structured method for defining, documenting, and maintaining requirements through the entire development life cycle. Typically, the software requirements focus on the factors that the software needs to address, without a detailed implementation assessment. On the other hand, the details of the software requirements depend on the organization, existing standards, and regulatory concerns [41].

2.3.2 SRS Development Process

Activities related to the software requirements development are visualized in Figure 11. Requirements development process can be described as reaching a conclusion on what the software should do and how it should function, and

Requirements Management as maintaining the same vision of the development process over time [42].

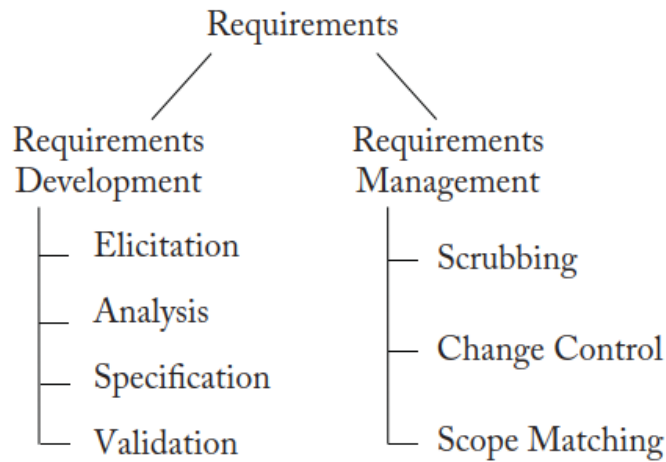


Figure 11: SRS Development Process [42]

2.3.2.1 Requirements Elicitation

Requirements Elicitation is the initial requirements engineering phase, which focuses on identifying and understanding the stakeholders' requirements for the software. Often, the elicitation process requires interaction with the stakeholders to capture their needs [43]. For a software project, typical stakeholders are clients paying for the software, customers deciding whether service the product, and users who interact directly with the software. One of the challenges relating to incompleteness of the software requirements can be associated with poor elicitation of the requirements. Furthermore, it can mean the inability to capture or acquire requirements from stakeholders [42].

Traditional techniques used for requirements elicitation include interviews, document analysis, questionnaires or surveys, and introspection methods. The goal of an interview is to verbally share ideas, information, and needs between stakeholders and analysts in a structured or unstructured manner. The results of the conversations are documented, and requirements elicitation is performed from the conversations. Document analysis involves reviewing and analysing existing materials and documents of the project or other sources to elicit requirements. Questionnaires and surveys can be used to gather requirements from a large group of people in case interviews or meetings are not possible. The contents of the questionnaires must be precise and well-planned to enable requirements elicitation. Also, the timeframe to answer the questionnaires must be mentioned. The Introspection method refers to the individual analysts representing their vision and thoughts on how the software should be designed [43].

Contextual techniques aim to elicit requirements in the context of the user. The activities of the end user are studied from the workplace, and requirements are gathered from the environment in which the software will be in the end implemented. Observation involves either passively or actively investigating end users' or customers' environments and taking notes on the different interactions of the user within the workplace. It allows the requirements engineer to obtain knowledge of what the users do, and it can help the engineer to write unambiguous requirements. Ethnography refers to a contextual method used to understand the organizational environment by interacting with different users and stakeholders. The goal of the ethnography is to determine the political environment of the organization [43, 44].

2.3.2.2 Requirements Analysis

Once the requirements have been gathered from their respective sources, a more detailed evaluation and analysis of the information elicited is needed to develop the requirements into a complete form [42]. The goal of requirements analysis is to understand, evaluate, and refine the information from the requirements elicitation to ensure that they are coherent and comprehensive before moving forward with the design stages. The detailed requirements analysis is an important phase of the development process, as, for instance, requirements communicated using natural language are prone to ambiguity, incompleteness, inconsistency, and incorrectness, which can result in misinterpretations, untestable requirements, untraced requirements to their sources, conflicts between stakeholders, and requirements. Furthermore, if the requirements tend to evolve constantly, the challenges intensify. Reaching an agreement on conflicting requirements requires communications and negotiations between the stakeholders, which eventually converge into a unified set of requirements [45].

To support the software requirements analysis process, general desired characteristics of each requirement are identified. These characteristics emphasize that every requirement is unambiguous (allowing only one possible interpretation), testable (fulfillment can be objectively confirmed), binding (meaning the client considers it essential and is prepared to invest in it), and atomic (addressing only one requirement at a time). Additionally, they need to capture the true needs of stakeholders, be expressed in their own vocabulary, and make them acceptable to all parties involved. After each requirement fulfills the previous properties, the set of requirements should be complete, adequately addressing boundary conditions, exception conditions, and security needs. They should be concise, containing no extraneous content, and internally consistent so that no requirement conflicts with another. In addition, they must be externally consistent, meaning they do not conflict with any source material. Finally, the requirements should be feasible, such

that a viable and cost-effective solution can be created within the given cost, schedule, staffing, and other constraints [42].

2.3.2.3 Types of Software Requirements Specifications

As previously described in 2.3.1, the requirements specification is the process of recording user needs and system functionalities in a structured document. The document contains complete user requirements, stakeholders' requirements, and system requirements represented as clearly, unambiguously, understandably, and consistently as possible. The specification typically distinguishes the requirements into functional requirements, referring to the statements of what the system must do to meet user needs, and non-functional requirements describing the limitations or conditions on the software and the development process due to organizational, product, or external constraints. The rest of this subchapter covers the different methods of carrying out the specifications and the general structure of the software requirements specification [46].

Natural language specifications are the most widely used method for specifying system and software requirements due to their intuitiveness, expressiveness, and universality. However, they are tediously prone to ambiguity and vagueness. To avoid misinterpretations in natural language specifications, the author of the specification is recommended to utilize a standard format for all requirements and emphasize a few short sentences. It is important to differentiate between mandatory and optional requirements. Mandatory requirements are often stated with "shall" and describe the requirements that the software must provide. Optional or desired requirements are expressed with "should". Depending on the audience of the specification, too technical phrasing should be avoided. Finally, adding rationales and using text highlighting in requirements helps to manage changes in the software requirements, as they can prevent making inappropriate changes. Structured natural language specifications use a more standardized technique for writing out the requirements in contrast to the unstructured text. The benefit of using structured specification is that it offers more consistency while preserving the expressiveness and understandability of natural language using predefined structures, such as templates. The specification can utilize constructs similar to programming languages to display iteration and alternatives. Despite the benefits of using structured language, it can still be difficult to express the requirements in a clear and unambiguous way, especially if complex computations are needed [46]. The structured natural language specifications increase the precision and conciseness. A simple instance of a structured form of requirements is the actor-action format, where the actor typically refers to the system, and the action is the function that the

system has to execute. The action can also be followed by an optional qualification [42].

In a Model-based requirements specification, the system is represented using structured modelling languages, such as Unified Modeling Language (UML) or Systems Modeling Language (SysML), to capture the system requirements in a technical structure. These modelling languages help to represent the system requirements accurately, reducing ambiguity found in natural-language specifications. Requirements models are categorized into Structural models, which specify the policies to be enforced, often also referred to as conceptual data models and logical data models. Behavioural models specify processes to be carried out, such as use case modelling, interaction diagrams, and state modelling. Also, the previously mentioned UML activity diagrams and data-flow modelling fall into the behavioural models. Model-based requirements specifications also vary in formality. As an example, Agile modelling tends to use sketches to communicate important information rather than using formal modelling notations. Semiformal modelling provides some level of modelling language semantics, however, these definitions may not be formally proved to be complete as well as consistent. In a formal modelling, such as specification and description language (SDL), there are strict mathematical semantics supporting mechanical analysis. A high level of formality can facilitate translation to code, aid in deriving acceptance tests, and reduce ambiguity. Finally, it is crucial to remember that formality may cause additional work and burden to the author and the reader. The issue can be alleviated using formal foundations with a more readable syntax [42].

2.3.2.4 Software Requirements Validation

Requirements validation aims to ensure consistent, complete, and conflict-free requirements. Requirements validation shares similarities with requirements analysis, but requirements validation is more concerned with confirming the correctness of documented requirements, while requirements analysis is more focused on understanding and documenting user needs [47]. Validation of requirements is an essential part of the software development, as inconsistencies and errors in the requirements can lead to significant rework and of the system and additional costs if the errors need to be fixed in the later stages of the development life cycle [46-48].

The consistency in requirements tries to achieve equivalent requirements components. Consistent requirements should follow the same naming and sequence of the requirements throughout the requirements specification. Also, consistency occurs when the requirements are accepted as satisfying the customer's needs. The requirements are considered complete when no missing definitions or constraints are present in the software system. In addition, requirements are correct when they are captured accurately [48].

Table 2, adopted from [48], describes the detailed descriptions for correctness, completeness and consistency.

Table 2: Correctness, Completeness and Consistency definitions [48]

Type of Requirement Quality	Description
Correctness	<ul style="list-style-type: none"> • Describes the correspondence of that specification with the real needs of the intended users in much the same way that correctness of a piece of software refers to the agreement of the software part with its specification. • A program is considered correct if it behaves as expected on each element of its input domain. • An SRS is correct if and only if every requirement represents something required of the system to be built.
Completeness	<ul style="list-style-type: none"> • Implies that all customer's needs will be met when the system is constructed. • A requirement must have all relevant components. • A requirement's document should include requirements that define all functions and the constraints intended by the system user. • It specifies required behaviour and output for all possible states under all possible constraints. • Responses of the software to all realizable classes of input data in all realizable classes of situations is included
Consistency	<ul style="list-style-type: none"> • No two or more requirements in a specification contradict each other and the case where words and terms have the same meaning throughout the requirement's specifications (consistent use of terminology) • Requirement uses terms in a manner consistent with their specified meanings. • Requirement should be understood precisely in the same way by every person who reads it. • Requirements in the document should not conflict. • Consistency is also referring to situations where there is no internal (logical) contradiction in a specification of a system. • Consistent specification exists when there is a computational model for its implementation, and the specification is valid when it satisfies the user requirements. • An SRS is internally consistent if and only if no subset of individual requirements stated therein conflict

The most profound techniques used in the requirements validation process are requirements prototyping, requirements reviews, requirements testing, and viewpoint-oriented requirements validation [48].

Requirements prototyping is a technique that aims to produce an executable model of the system and share it with the end-users and customers. They experiment and test the system, providing feedback and refinements to the requirements [46, 47]. Two types of prototypes are generally used in the requirements validation process. Throwaway prototypes are intended to be discarded after the initial requirements have been agreed upon and built into the prototype. They support resolving conflicts in the requirements between the development team and the customer. Once consensus is reached between the parties, the requirements are incorporated into the SRS, and the prototype is discarded. In contrast to throwaway prototyping, evolutionary prototyping utilizes a small set of stable, quality-driven requirements to create a prototype of the system. The prototype is continuously improved through user feedback. Prototyping allows customers to obtain visual information about the software system development, identify issues with requirements, and provide additional requirements if necessary. The downside is that prototyping can be a time-consuming process, creating additional costs and adding uncertainty if the prototypes can not be transformed in to executable version after validation [47].

Testing-based requirements validation process constructs testing artefacts that aim to validate the software system requirements indirectly. In case the generation of such artefacts is considered easy, the requirements are high-quality [49]. The advantage of using requirements is that it can eliminate the unnecessary requirements, in addition, the generated test cases can be possibly used in the future for the complete testing of the ware. Large software organizations with skilled testing teams benefit from requirements testing, however small organizations with less experience in test case generation, the requirements testing can cause additional costs [47].

One of the most common ways to validate requirements is to use requirements reviews or inspections. In a requirement review, one or more reviewers are asked to find errors, invalid assumptions, lack of clarity, and deviations from accepted practice. The reviewer is guided in the process by using checklists to focus on specific aspects of the requirements. The checklists can contain quality criteria or so-called “definition of done” to help them focus on different aspects of the requirements specification [42].

Model-based requirements validation builds formal representations such as Bayesian, conceptual, and goal-oriented models in order to evaluate the requirements correctness, quality attributes, dependencies, and variability. This method allows early detection of inconsistencies between the requirements and design, as well as facilitating the interpretation of the domain and enabling simulation when prototyping is challenging due to safety-critical aspects. Problems tend to arise from reusability issues, scalability, and

difficulty determining which quality factors to evaluate, thus limiting the ability to uncover exact errors. As a result, model-based validation should be combined with testing or knowledge-based techniques for complete validation [49].

2.4 Lifecycle Models in Safety-Critical Environments

A software life-cycle model (SLCM) provides a conceptual framework that describes the structure and sequence of processes, activities, and tasks to be performed throughout the software system's life cycle. It represents the evolution of a system or product from conception through retirement, covering both the technical and management processes required to meet the stakeholders' needs [42]. This section studies the most used software life cycle models (SLCM) and their use in safety-critical environments. The advantages and disadvantages of each model are discussed, in addition to the concerns imposed by the safety-critical environment and the development of safety-critical software. Further, this section covers the use of the Waterfall model, V-model, Iterative and Incremental model, and Agile methodologies.

The traditional lifecycle models used in the safety-critical systems cover models such as the Waterfall, V-model, and Iterative and Incremental models. These SLCMs have been able to demonstrate the safety of the product successfully [50]. Traditional SLCM methodologies have proven to be valuable and useful in safety-critical software development due to the heavy documentation required to meet the regulatory process standards and requirements, which is also emphasized in traditional methodologies. Further, the traditional methodologies, such as the Waterfall model, tend to be successful when the risks and requirements are well understood in advance, leaving little room for changing requirements later in the project [51].

The Waterfall model is a sequential software development model, where each individual phase needs to be completed before moving to the next phase. It is generally easy to understand and implement, and tends to be effective in small projects. The Waterfall model is typically constructed from six different fixed sequential phases. At first software requirements specification document is created, and all requirements are documented. Next, the software is designed based on the requirements specification. This phase is followed by the implementation phase, when the actual development of the software is executed. The implementation is followed by an extensive testing phase, where all software units are integrated, tested, and bugs and defects are fixed. After implementation and satisfaction of non-functional and functional requirements, the software is tested, validated, as well as deployed in the customer's environment. The final maintenance phase covers the issues raised in the customer's environment, including small enhancements, in case the user is not completely satisfied [52]. V-model, also referred to as Verification

and Validation Model, follows a similar sequential process as the Waterfall model, while extending the development phase to a corresponding testing phase. This ensures that each stage of development is verified and validated before proceeding to the next stage. The V-model has a strong emphasis on detecting the problems and defects early on with extensive testing parallel to the development. The model is widely used in the safety-critical software development, when the requirements are well elicited early, ensuring high quality and comprehensive testing coverage. Traceability is assessed by tracing each design element requirement to a corresponding test case. Issues often arise from changing requirements, as the changes can be difficult to adapt to the sequential phases. Also, the user involvement after the requirements phase is low and can easily lead to a misunderstanding of the expectations [53].

The iterative software development process model consists of repetitive cycles of development, where each iteration adds a feature or improvement to the previous software version based on user feedback and changing requirements. This model is effective in software projects, where the requirements are prone to changes during the project. The iterative model is a flexible model that comes with various advantages, especially from the focus on quality and customer satisfaction, as the defects can be detected early in the project. The risk management process is also a vital part of iterative development. When the risks are addressed iteratively, the potential of major issues being introduced later in the project is reduced. In contrast, the documentation overhead required for each iteration can cause additional costs and resources. Also, the repeating cycles can cause extensions to the project schedule. In the Incremental Model software development process, the software is developed in a project that is divided into different increments, each of which contains a part of the software's overall functionality. When the increment is completed, it will be implemented in the current version of the software, which will eventually develop into the complete software product because of the increments. The incremental development process suffers from similar problems as the iterative approach, while the model emphasizes early feedback, it may not be available to the stakeholders until later in the development process. The incremental process model tends to work well with large projects, where the development process can be divided into clear milestones [53].

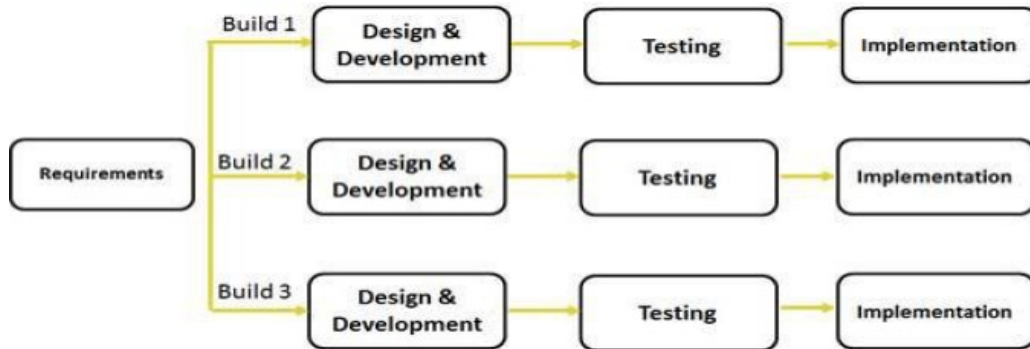


Figure 12: Iterative Development Approach [52]

With the need to deliver high-quality software systems faster and at a lower price, Agile methodologies can offer an alternative approach despite the various contradictions in key areas regarding safety-critical systems [50]. These issues focus on the documentation, flexible requirements written in user stories, iterative and incremental lifecycles, as well as the Test-first process. The documentation is one of the main challenges for adopting agile software development for safety-critical products. As the safety-critical software development demands evidence that all processes demonstrate sufficient quality and safety requirements, in contrast, agile software development does not see maintaining the documentation as an essential area. It is guided by principles such as ‘just enough’ or ‘barely sufficient’ and focuses on delivering functional software rather than comprehensive documentation. This can cause problems as the regulatory processes and requirements are fundamental pieces in achieving high quality. Additionally, fear of regulatory inspections and compliance-related concerns are raised as the minimal documentation can be insufficient to determine the quality of safety-critical systems during regulatory agencies' inspections. The second problem recognized is that the agile processes use loose requirements, such as flexible requirements and user stories, which are written by customers, possibly in business language, while safety-critical development processes discourage the requirements changes due to additional costs and work required in redesigning, testing, and documentation. Another challenge arises when the agile methods encourage the use of iterative and incremental steps, which include all development phases. Further, each iteration should result in a validated and tested software product that the customer can use [51].

Despite the challenges related to the safety-critical software development using Agile methods, various tailored hybrid models from well-recognized Agile SDLCs have been researched, such as Safe Scrum or Scrum for Safety and Agile V-Model. The concept of the hybrid development approach combines agile and traditional plan-driven approaches and tailors them to the organization's own needs, for instance, processes, or specific projects. Hybrid approaches in regulated domains benefit from the flexibility of agile

approaches while maintaining compliance with standards and regulations typically emphasized by the plan-driven approaches [54]. Agile V-model proposed by McHughes et al. utilizes V-Model as a plan-driven life-cycle model, while incorporating agile practices into the process. The idea is to combine V-Model, widely used in regulatory environments, especially in medical device software, and identify agile practices that apply to the regulatory standards and guidelines. The proposed model, with identified practices shown in Figure 13 [55, 56]. Later, Khan et al. proposed an enhancement to the model by introducing a plan-driven window used for safety-critical, regulatory, and high-risk fixed requirements in addition to an agile window, which can handle the changing as well as evolving requirements. In this approach, reapproval is mandatory only if changes occur in the safety-critical regulatory requirements. The enhancement has the possibility to reduce time to market, however, it also adds complexity to handle different development windows, as well as additional planning is necessary [57].

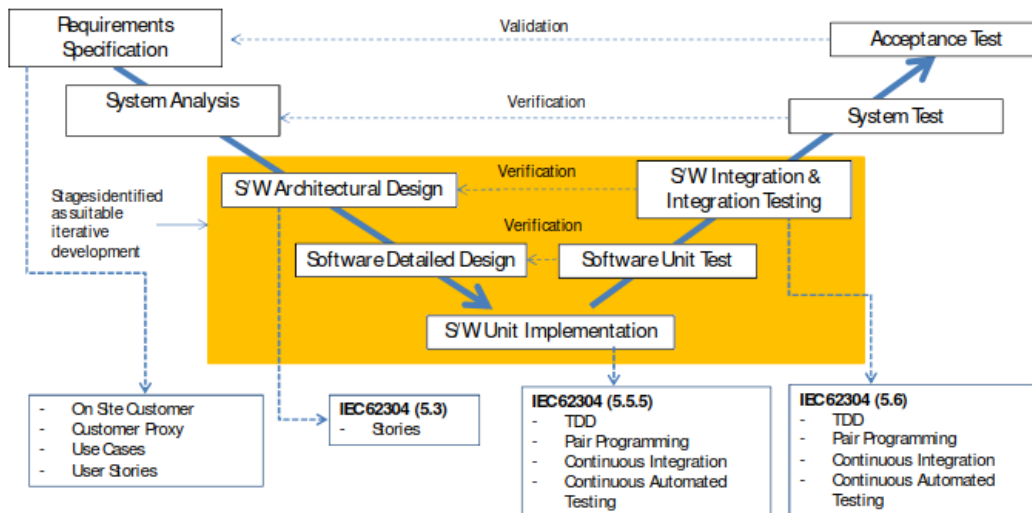


Figure 13: Agile V-Model Concept [55]

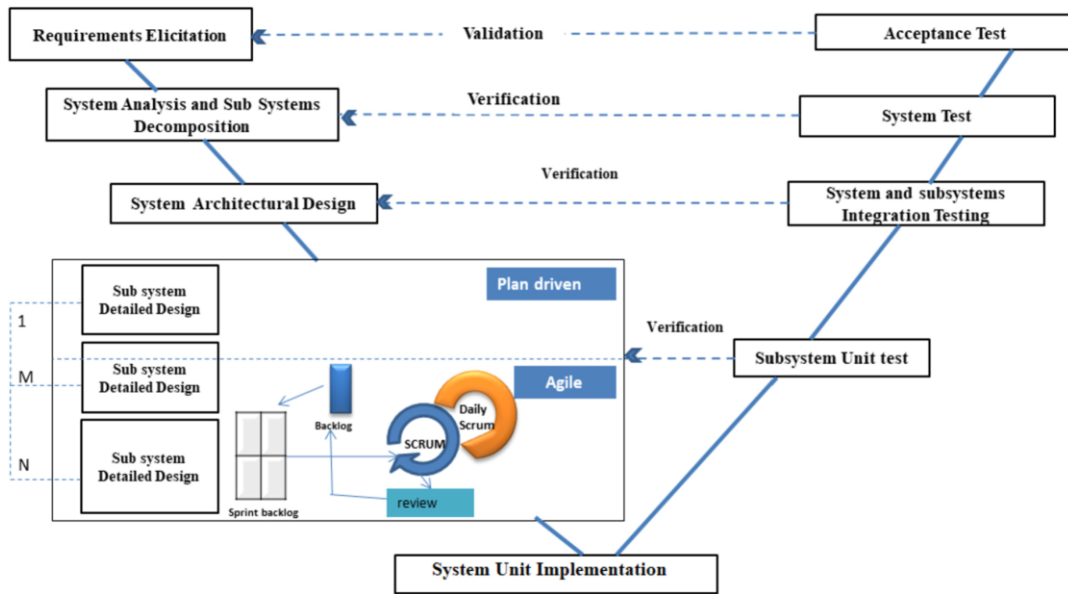


Figure 14: Enhanced Agile V-Model [57]

The Scrum for Safety framework utilizes Safe-Sprints, where each increment will provide documentation as well as conformance proof. It integrates the verification and validation activities throughout the sprint at the same time while updating the documentation incrementally. Further, the requirement-implementation links are tracked in real-time. While adopting the benefits of agile development, the documentation overhead can still be an issue if most of the time in a sprint is devoted to the verification and validation activities, as identified by Barbareschi et al. [58].

3 Research Material and Methods

This chapter outlines the traceability system framework, risk management, and requirements engineering methods used to develop the desired traceability system, as well as the Software Requirements Specification (SRS). The development process of the system is further covered in Chapter 4.

3.1 Traceability Assessment Methods

Materials and methods used for the traceability assessment phase include in-depth document analysis of the manufacturing specification documentation. This document contains detailed information about the manufacturing process phases. In addition, product design specification documents are studied to gather a thorough understanding of the product and manufacturing. The tools and methods used to achieve a traceable system are covered by the literature review. The author is responsible for ensuring each requirement of the proposed software is traceable to the relevant documentation and supports the end-to-end unit-level traceability of the manufacturing process.

Requirements traceability matrix (RTM) covered in 2.1.3 is used to trace individual requirements from PFMEA documentation, stakeholders, other environmental constraints, and other sources at a more detailed level. In addition, it allows clear bidirectional traceability, as highlighted in 2.1.2, between software requirements, as well as their origins. RTM is effective, especially in cases where the number of requirements and sources is relatively low. However, RTMs can be difficult to manage if the number of trace links is high [7]. Although RTMs can become hard to manage in complex systems, the scope of the system made the approach suited for ensuring the structure, traceability, and completeness of the system.

PFMEA risk management method, which was covered in 2.2.3 and is included in 3.2, offers a variety of options for traceability. The requirements that the software needs to address can be traced to the specific rows with unique identifiers in the PFMEA worksheet, establishing traceability between the risk management process. If the software acts as a control method or improvement action, as in Figure 10, it can be traced back to the relevant software design artefact, verifying that the developer implemented the appropriate controls. Additionally, PFMEA method is used to cover the traceability between product design using numerical references to the product Cause and Mitigation Table (CMT).

The IoT-enabled traceability covered in 2.1.6.4 was chosen for the system framework. IoT is a proven concept to increase traceability, while ensuring authenticity and supporting tracking the origin and quality of the product, with real-time tracking capabilities [24]. Further, IoT, OPC UA, and Modbus TCP/IP communication-enabled data capture was incorporated by using

manufacturing equipment with IP-based interfaces and barcode scanners as IoT nodes in the system. These devices captured process and batch data in real time, such as printing parameters, curing profiles, and thickness measurements, and transmitted them into the centralized traceability system.

Another benefit of using an IoT-based approach is that it is naturally emphasized by implementation with cloud infrastructures. This aligns well with the existing eDHR cloud framework used by the company. Although there is a lack of standardized frameworks to support DPPs as stated in 2.1.6.2, some fundamental concepts were incorporated into the traceability system design. The approach emphasized digital records of the product in addition to the origins and the material decomposition of the product throughout the manufacturing process. Also, as proposed by Hendro Wicaksonoa et al., the IoT and physical identifiers are utilized by the conceptual DPP architecture for data collection at the first tier, further validating the IoT-based approach and aligning with the MES real-time data collection practices.

Both indirect part marking and direct part marking techniques were utilized by the traceability system. A unique QR code is engraved into the surfaces of the product by utilizing laser equipment, enabling unit-level traceability. Laser-engraved QR code is specifically effective, and due to its durability and applicability on small parts, making it applicable for product identification [3, 25]. To enable seamless operation and compliance with the product identification, the materials used in the process were marked using indirect marking QR code labels, which are attached to the surface of the material and can be read using a camera-enabled barcode scanner. The 2D barcodes are generally preferred in medical devices as they have a larger data storage capacity, and they can be implemented into smaller surfaces as well as support omnidirectional readability, which is essential when handling manufacturing materials, such as in this case [25].

Success was defined as the ability to automatically and reliably associate these captured IoT events with corresponding unit records in the eDHR. Cloud-based eDHR validation was performed by implementing a cloud repository for electronic Device History Records. Validation was carried out in controlled test builds iteratively, which were excluded from customer shipments, to confirm that IoT, OPC UA, and Modbus-generated events and operator inputs were correctly reflected into the eDHR with traceability links from the batch, process, materials, and equipment.

As a consideration of limitations regarding the methods, the printer's PLC did not support communication through OPC UA natively. Changing the PLC would have resulted in additional costs. However, the printer did not support the Modbus TCP/IP by default and had to be reconfigured similarly as illustrated in Figure 5. Also, KepServerEX could have been used as a middleware to transfer the Modbus TCP/IP connection to OPC UA format, however, this approach would have required running the KepServerEX software constantly in the background, which would have added additional steps. The

Requirements Traceability Matrix (RTM) was structured to focus on essential requirement origin relationships, ensuring clarity and maintainability without adding unnecessary complexity, while mapping the requirements to adequate test cases. The possible issues regarding the IoT devices' availability of data were handled by designing to poll the equipment for data at specific events, more specifically, during a barcode scan or manual operator input.

3.2 Risk Management Methods and Materials

The risk management methods used in this thesis is done by literature review in addition to document analysis of the design verification risk analysis of the product. The design verification assesses risks by utilizing the PFMEA method. This structured method is widely used in the manufacturing of safety-critical products, such as in medical device manufacturing, as previously discussed in 2.2.3. Also, the PFMEA method supports the company's existing quality management processes. It helps to avoid the additional workload required to meet the company's guidelines and regulatory compliance. Another benefit of using PFMEA is that it enables the potential to support traceability, which is one part of the thesis and part of the SRS development and evaluation methods. The PFMEA process steps follow a structure like Figure 9, with small modifications due to the workflow. The notable difference between Figure 9 and the process adopted to the project is the iterative nature of the manufacturing testing. Step 5: Review and update the FMEA is done by reviewing the feedback from the test builds, and the FMEA is updated accordingly in case new failure modes, changes to the RPNs, or new mitigation strategies are introduced during the test builds. After the test build iterations and acceptable RPNs are reached, the FMEA is frozen for manufacturing validation, and a formal approval process is executed.

GE HealthCare has strict internal guidelines that need to be followed in the medical device process risk management. This thesis follows the company's risk management policy. The benefit of existing guidelines helps to comply with the international standards and regulations set for the medical device manufacturing process. Further, in this thesis, the company's internal guidelines are not discussed in detail, however, the writer of the thesis is responsible for using the guidelines to achieve compliance with the risk assessment. These guidelines that concern the risk management operate under the ISO 13485 global Quality Management System standard that applies to the PFMEA process.

The development methods of the risk management include interviews, structured meetings, where the scope of the meetings is predefined to maximise the effectiveness of the meeting, and reviews with company professionals of different backgrounds to ensure documentation is aligned with the company's guidelines. In addition, in-depth individual research focusing on document analysis of the existing manufacturing specifications, design risk

management processes, and equipment is conducted. The success of the Risk Management phase was defined as the ability of the proposed software to mitigate the high-level risks identified from the PFMEA to an acceptable level.

3.3 SRS Development and Evaluation Methods

The methodology for developing and evaluating the Software Requirements Specification (SRS) is based on established software and systems engineering standards, industry best practices, and company-specific processes. The approach combines structured requirements engineering methods with risk-based inputs and traceability techniques to produce a clear, complete, and regulatory-compliant specification.

The key conceptual and practical knowledge for a structured requirements engineering practice is derived from The Guide to the Systems Engineering Body of Knowledge (SEBoK) and The Guide to the Software Engineering Body of Knowledge (SWEBoK v4.0). Further, ISO/IEC/IEEE 29148:2018 Systems and software engineering - Life cycle processes - Requirements engineering standard was applied to define the structure and content of the SRS. This standard was selected because it is widely recognized in regulated industries and aligns with good practice in both documentation and traceability. The remainder of this chapter covers the SRS development methodology, incorporating the activities suggested in Figure 5.

The SRS requirements elicitation process included various input sources covering documents and environmental design constraints specific to the screen-printing oven, curing, and thickness measurement production phase at GE HealthCare Finland. The PFMEA worksheet provided the foundation for risk-based design inputs. These design inputs were analyzed and prioritized based on their RPN values and divided into functional and non-functional requirements. Other relevant requirement sources included product manufacturing process specifications and blueprints, which contained detailed information about the product tolerances and manufacturing requirements. In addition to the requirements elicited from the previously mentioned sources, common techniques such as interviews, meetings, and brainstorming were used to elicit requirements, which were also identified in SWEBoK. As stated on 3.1. RTM was used to establish traceability between different artefacts. In more detail, this approach was used to link individual requirements to PFMEA risk items, as well as highlight the origin of other relevant design inputs and requirements elicited.

The analysis phase focused on refining the elicited requirements to ensure they were clear, feasible, and consistent. Each requirement was evaluated against key quality properties such as unambiguity, testability, feasibility, and stakeholder relevance. The complete set of requirements was reviewed for completeness, internal and external consistency, and feasibility.

Completeness checks verified that boundary and exception conditions, as well as safety and regulatory needs, were adequately covered.

The requirements are carried out in the software requirements specification using structured natural language. Structured natural language was chosen as it is often related to increased precision and consistency of requirements. [42] This approach also ensures that the requirements are human-readable, while supporting traceability and validation processes. Requirements were documented according to the structure recommended by ISO/IEC/IEEE 29148:2018, including separate sections for functional, non-functional, and interface requirements. The company-specific software development processes and quality management guidelines were followed to ensure compliance with internal standards and regulatory needs. The company does not provide a ready-to-use template for the SRS, however, the development process after the software requirements specification must follow a strict sequential document process. SRS was designed to act as a formal input to the company's design and verification lifecycle.

The validation phase ensured that the developed SRS met defined quality and regulatory objectives. Validation activities included periodic reviews with the software engineer and reviews of the system in the actual production environment with the stakeholder, consistency checks, as well as cross-references to PFMEA-derived risk controls. In addition, prototyping was used to create a functional prototype of the software, which was tested using test builds, while the software requirements and the system were enhanced based on the feedback. Each requirement was reviewed for traceability to its source, ensuring that all high-priority risks identified in the PFMEA were addressed through at least one corresponding software requirement.

The SRS development process and the software system development adopted a hybrid software development lifecycle combining the iterative software life cycle with the sequential Waterfall model. A hybrid approach can be tailored according to the project's needs, enabling flexibility without compromising compliance. [54] An iterative life cycle without the need to validate the software between the builds was made possible by the nature of the manufacturing process validation. The manufacturing process was tested and developed to its final form using test builds, which were excluded from shipping to the end-user. Implementing software to an already validated manufacturing line would have required a sequential life cycle approach, such as the Waterfall model. The software validation process in GE HealthCare requires a sequential approval process of the documents after an acceptable version of the SRS and the software has been developed. As a result, the iterative approach emphasizing functionality over completeness at the first test build was possible. The functionalities added to the next test builds / iteration were decided based on feedback from the software, high-priority risk-based requirements, as well as functionalities vital to efficient manufacturing operations. This method adopts fundamentals from the approach proposed by

Khan et al., where a plan-driven window is used for safety-critical, regulatory, and high-risk fixed requirements and an iterative window for evolving requirements [57]. This approach ensured that the software was more complete before the plan-driven validation process was required. The manufacturing test builds were used as iterations to refine the requirements and the software. This allowed the software to be more complete when a formal sequential validation of the safety-critical, regulatory, and high-risk requirements was required.

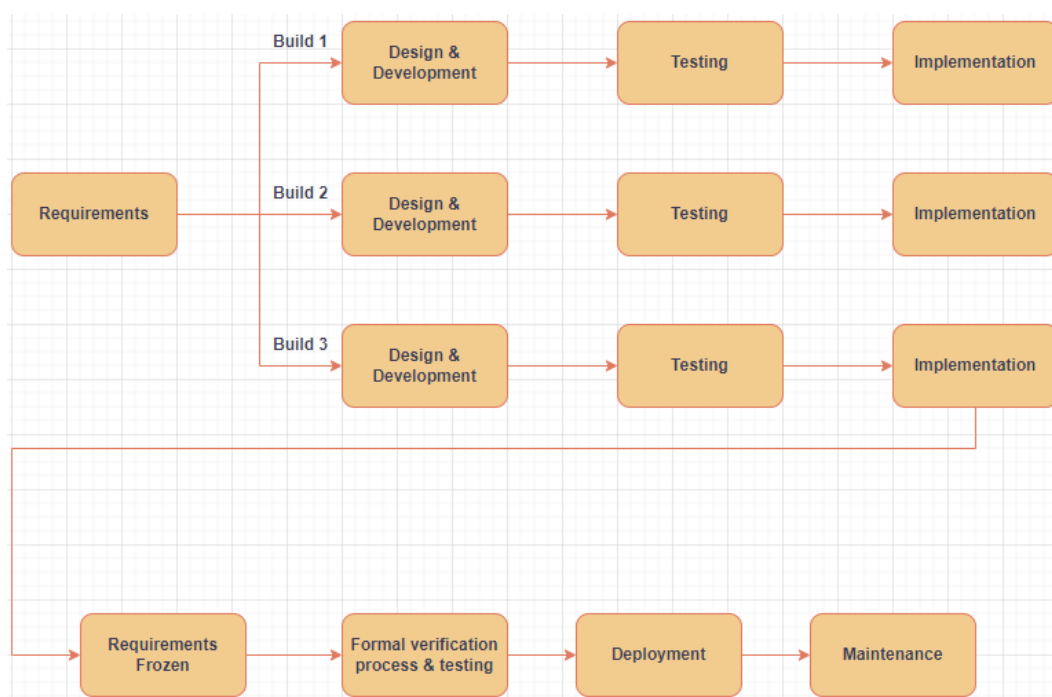


Figure 15: Life-Cycle Development Process

3.4 Documentation and Visualization Methods and Tools

Documentation and visualization are essential as they ensure clarity, consistency, and traceability throughout the thesis and to other parties involved in the project. They support fluent communication between stakeholders and enable paths to receive feedback. The results and progress of the thesis and of the thesis can be monitored if different concerns arise during the project. This section describes the documentation and visualization methods used to comply with the thesis requirements and provide transparency to the author's work.

Throughout the development process, a variety of documents and worksheets were produced to capture design inputs, requirements, and traceability information. Standardized templates were applied to maintain uniform structure and facilitate reviews. Flowcharts and tables were used to visualize

workflows, process logic, and requirement relationships in a clear and auditable manner.

For requirements traceability, a dedicated Requirements Traceability Matrix (RTM) was maintained. This matrix linked requirements to their corresponding PFMEA items, design constraints, standards, and test cases. Visualization of requirement hierarchies and system behaviour was supported by draw.io diagrams, which were integrated into the SRS to illustrate relationships between subsystems and process phases.

Company-specific systems and tools were also utilized to support documentation and data visualization. These included eDHR cloud services for storing and verifying electronic Device History Records. In addition, UA Expert and KepserverEX were used for testing and visualizing OPC UA-based communication between equipment and the software, and company-provided templates to ensure compliance with internal quality management and regulatory requirements.

4 System and the Software Requirements Specification Development Process

This chapter outlines the development of the traceability system and the construction of the Software Requirements Specification (SRS) created as a part of GEHC Finland's design transfer project. The purpose was to design a production monitoring and data collection system framework, as well as an SRS of the software responsible for controlling the system, transferring unit data records automatically to cloud-based storage, and removing the manual collection process to improve the efficiency of the process. The system needs to support end-to-end unit-level traceability, risk management, and real-time data collection during medical device manufacturing.

4.1 Development Process Overview

Medical device manufacturing is a strictly controlled and regulated domain, both nationally and internationally, as even small abnormalities or changes in the manufacturing can impose risk to human health. As a result, the manufacturer must be able to demonstrate that sufficient controls are in place to ensure the safety and reliability of the product. A structured approach is needed to verify compliance, traceability, and risk coverage of the manufacturing systems. The systems must provide end-to-end traceability from requirements to post-market surveillance in case of recalls.

The scope of the system covered by the system and the SRS in this thesis is a part of the screen printing, oven curing, and layer thickness measurement process. The proposed software gathers specified information from the printing process to the product's electronic device history record (eDHR).

The remaining chapters present the system and SRS development process used to establish traceable and hazard-free automatic eDHR software for the screen printing, oven curing, and thickness measurement manufacturing phase. The development follows a risk-driven and manufacturing specification-driven requirements elicitation and analysis approach, which is covered in Chapter 4.3. The proposed system framework development process is covered in Chapter 4.2, highlighting the traceability framework used. The SRS development process is introduced in Chapter 4.3.

The project adopts a hybrid development model. Iterative approaches in the safety-critical environment are often challenging as the software implemented in the production environment must be fully complete and validated. However, the nature of the design transfer project allows the use of incomplete and unverified software to be deployed in test builds prior to mass production. These builds are used only in unofficial production units and are not released to end customers, thereby allowing iterative testing, requirements change, and refinements without compromising safety or compliance. The

benefit of using an iterative approach in the requirements is that key safety-critical requirements can be elicited and approved early. If necessary, functional and usability requirements can be improved iteratively from the test builds, feedback, and incorporated into the SRS before final verification and validation of the software and SRS.

4.2 System Development Process

The system development process starts by analysing the process as a whole and deriving the scope of the system. To accomplish these requirements, the system requirements were gathered from internal meetings, process architects, manufacturing specifications, design inputs, and PFMEA. The analysis concluded that the system scope is defined to cover the manufacturing process from the screen printing, oven curing, and layer thickness measurement phases.

The product identification was developed using a direct part marking approach and implemented by using laser engraving equipment. The equipment software, which uses ladder-like programming language, is configured to create the identification QR code identifiers on the surface of the product containing specific data. This allows the rest of the system to attach data to a specific unit based on the QR code at any specific manufacturing process. After the requirements review of the correctness of the code data, it was formatted to contain the manufacturing date, part number specific to the product, two letters describing the site identifier, revision, and a sequence number separated by a special character. The sequence number is incremented after each laser cycle and resets at the beginning of each manufacturing day. This way, the physical product can be identified by the software covered in 4.3 by parsing and combining the laser engraving date with the running number. This product does not require an official UDI direct part marking on the surface of the product because the product is not meant to be reused or reprocessed, as discussed in 2.1.6.4. However, the marking structure incorporates UDI-PI elements, such as manufacturing date and serial number for internal traceability, specifically to enable unit-level traceability, instead of lot- or batch-level identification. The product identifier is shown in Figure 16. Finally, the readability was validated by using a camera-based QR code reader specified to be used in the manufacturing process.



Figure 16: Laser Engraving Direct Part Marking

The Figure 16 QR code contains the data in a format as shown below in Figure 17

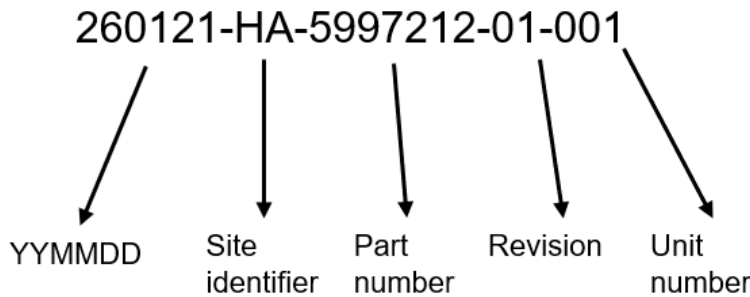


Figure 17: QR Code Contents

Material identification was incorporated by using indirect part marking methods on the surfaces of the materials, such as inks, which needed to be exposed to tracing. 2D barcode labels, more specifically Data Matrix, was created and attached to two separate locations on each part to mitigate the possibility of loss or damage to the Data Matrix. To enable identification of the specific material, the code contains the part number, LOT, and expiration date, separated by a special character to enable parsing the data in the software. An example labelling is shown in Figure 18, but due to confidentiality of the inks used in the process, more details are not discussed.

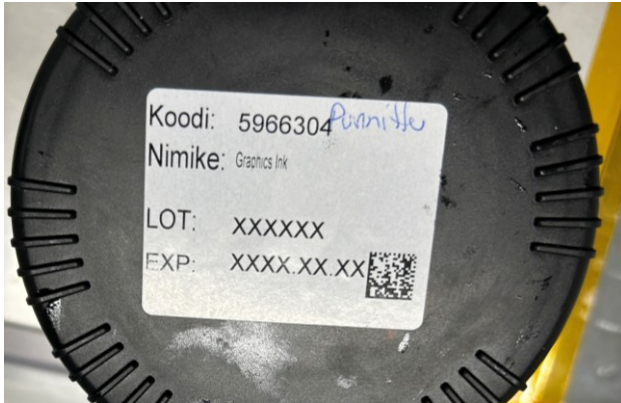


Figure 18: Laser Engraving Direct Part Marking

Similarly, as for the part identification, the readability of the codes was validated by using a camera-based QR code reader specified to be used in the manufacturing process. In addition, indirect part marking was utilized to identify the printing screens used in the manufacturing process. RFIDs were considered to be used for identification, however, the screen tensioning is outsourced, which contains a high risk of losing the RFID identification in the process. As a result, a company-specific identification was applied to the surface of the screen frames in-house.

A centralized eDHR cloud structure for the product was created to document the product's manufacturing sequence. This structure contained rules for predefined process steps, so that the product can only proceed to the next manufacturing step once the previous step has been completed. Initial DCPs (Data Collection Plans) were integrated into the eDHR workflow, which allowed manual data inputs and collection from early test builds throughout the development process, even without the software implementation. The cloud-based storage also lays the foundation for IoT integration. As discussed in 2.1.6 cloud-based approaches are crucial for IoT-enabled traceability, supporting high volumes of data and scalable data storage. Figure 19 shows the proposed eDHR operations, as well as the required DCPs for each operation. The operations refer to the individual phases required to complete the process step. Rework operation was included in the logic to handle failed results from thickness measurements and scraping, in case the product was damaged in the process. These DCPs form the foundation of the digital records created by the system. The actual completeness and integrity of the audit trail are further represented in the results in Chapter 5.3 regarding the DCPs 1-7. Further, the more detailed representation of the software logic is presented at 4.3.2.

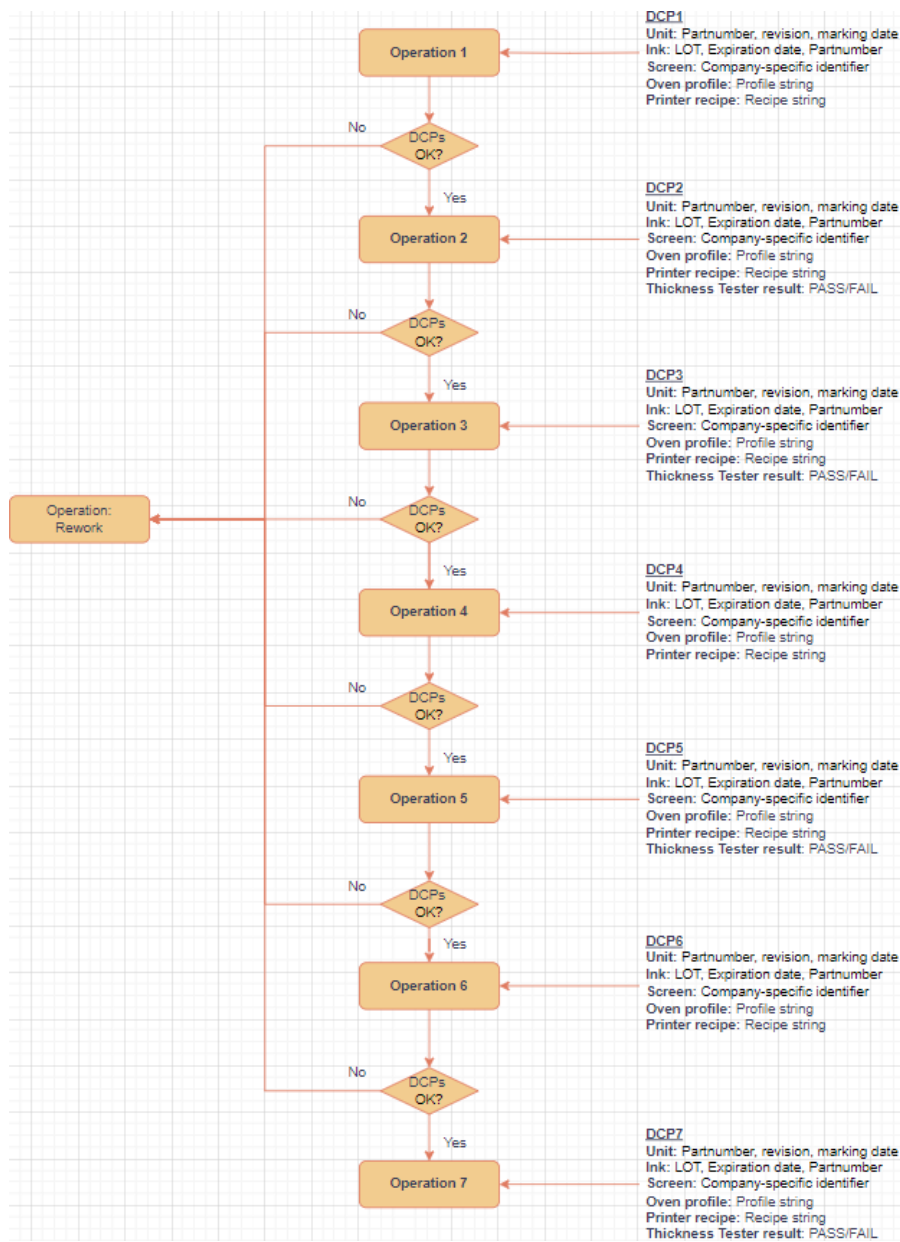


Figure 19: Flow Chart of The Proposed eDHR Structure.

The IoT Architecture utilized a 3-layer approach similar to Table 1, including Application Layer, Transmission Layer, and Perception Layer. The perception consists of the barcode scanners and PLCs associated with the oven and printer. The Transmission Layer contains communication technologies and protocols such as Bluetooth, USB/Ethernet adapters, internal factory LANs, OPC UA, and Modbus TCP/IP. The Application Layer includes the central PC software, Line Control Software, and the eDHR Cloud platform, which uses the Machine Connectivity Interface to communicate with the equipment.

The manufacturing equipment's PLCs provide the perception layer of the IoT framework architecture stack. These equipment are embedded with sensors allowing real-time data acquisition through M2M communication protocols at the connectivity layer, and the traceability application discussed further in 4.3 represents the application layer. The flow oven equipment supports OPC UA M2M communication natively, enabling easy access to all of the controlled parameters at the machine. The data collection plan and critical nodes were identified based on the FMEA risk management process. These included the curing profiles as well as the status of the oven. As the status node indicates from the internal sensors whether the oven has reached the acceptable production parameters, it is enough to monitor this node. Further, the profile refers to the current settings set for the oven based on the production process. It is essential to ensure each profile is used at the correct layers to prevent under-curing and causing damage to the product. The UA Expert software was used to demonstrate and validate the correct nodes, which are displayed in Figure 20.

#	Server	Node Id	Display Name	Value	Datatype	Source Timestamp	Server Timestamp	Status
1	SIMATIC.S7-1200.	NS4 Numeric 2980	Recipe Data Record - Dual Cure	UV Cure	String	12.29	12.29	Good
2	SIMATIC.S7-1200.	NS4 Numeric 1326	Dryer Status Line	6	UInt16	12.29	12.29	Good

Figure 20: Oven Real-Time Data Collection Validation

The printer required configuration of the PLC software in order to provide real-time data for traceability and risk mitigation needs. An approach similar to Figure 2 was used to configure the PLC as a Modbus Server. The PLC reads the sensor value from the analog inputs and stores it in the Modbus register, which can then be read by creating a Modbus client capable of reading data from the specific registers in real-time. Figure 21 shows the approach used for the Modbus configuration.

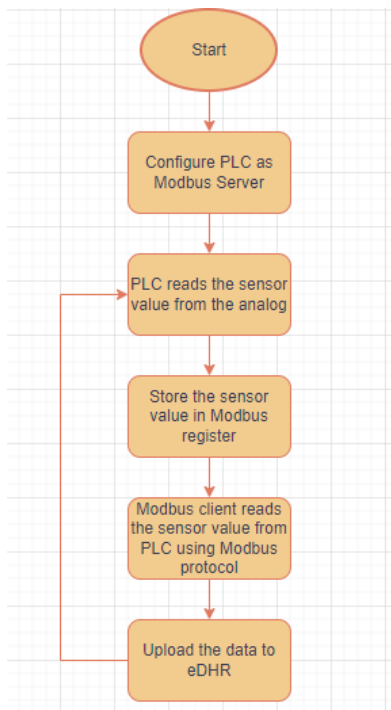


Figure 21: Modbus Configuration Process Flow

In practice, the configuration was done by using Mitsubishi GX3 software to pull the software from the equipment by doing a full read from the PLC. Next, the Modbus communication was modified in the software as shown in Figure 22.

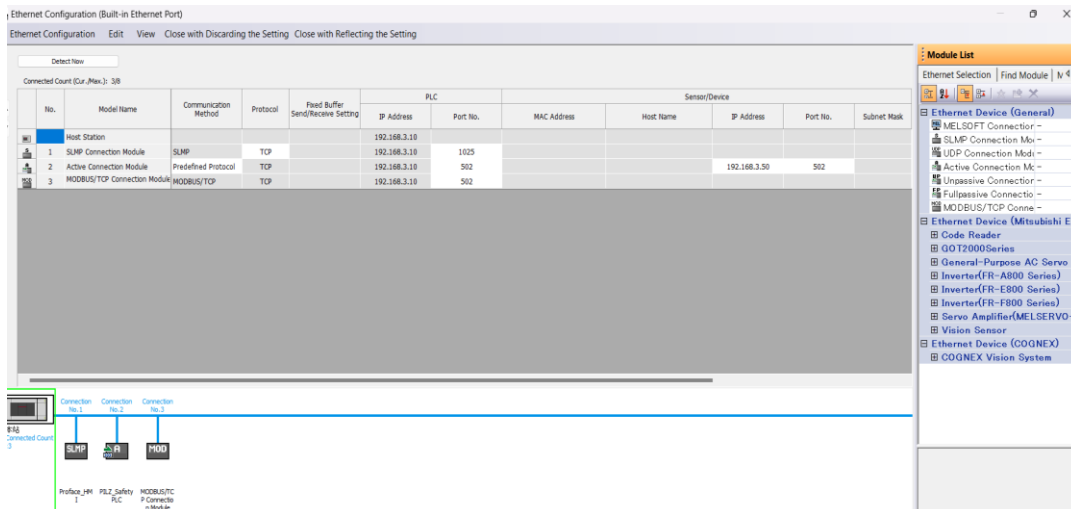


Figure 22: GX3 PLC Modbus Configuration

After modifying the PLC software Ethernet configurations, the software is downloaded back to the PLC to use the modified software. Similarly, to the oven, the critical Modbus registers needed for tracing were identified from the FMEA, and the data connection, as well as the parameters, were validated by using KepServerEX as a Modbus client. The FMEA requires a mitigation

that each operation, must be completed by using specific equipment settings that can vary between the operations. These registers contain the settings used in real-time, which were validated by using KepServerEX OPC Quick Client in Figure 23

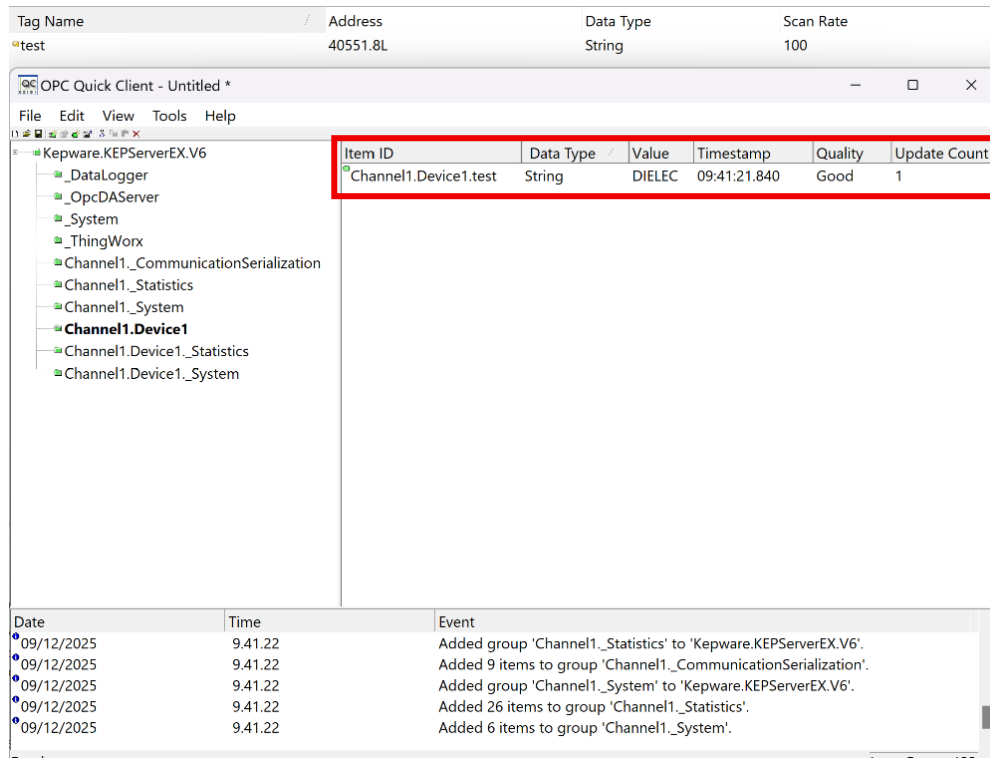


Figure 23: Printer Real-Time Data Collection Validation

Other tasks needed for the traceability system framework included configuring the tester pc used to send data to a specified company network drive, which was later used to validate the tester results at the central PC. Further, the central PCs Ethernet ports were configured with static IP addresses to communicate with the oven and the printer. The successful communication was verified with the ping command to ensure the company's firewall did not interfere with the connections. Finally, a barcode scanner with Bluetooth capability was inserted into the system. Figure 24 visualizes the proposed system architecture.

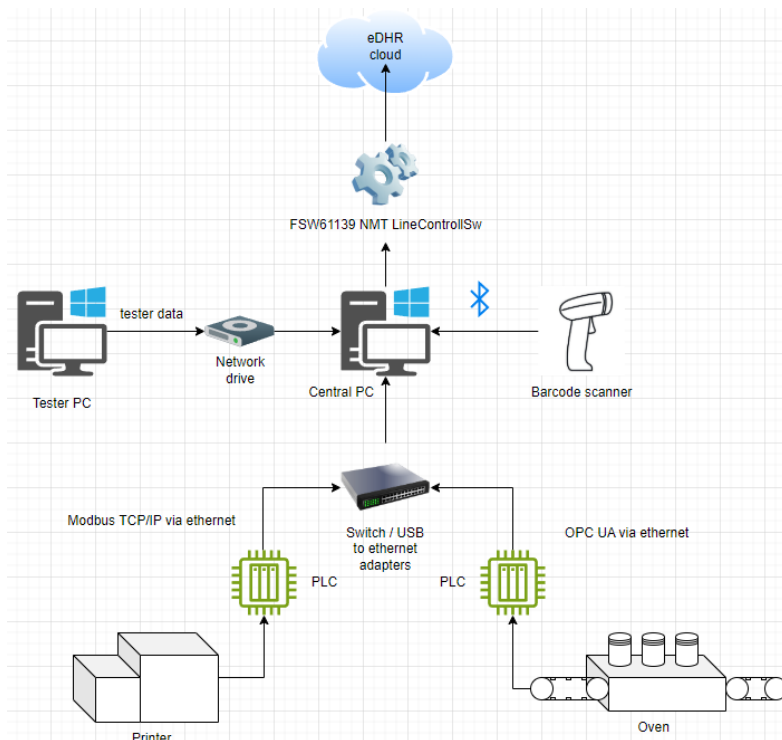


Figure 24: Proposed System Architecture Diagram

4.3 SRS Development Process

The Software Requirements Specification (SRS) development follows a structured approach, similar to the 2.3.2. including requirements elicitation, analysis, specification, and validation steps. Sections 4.3.1 and 4.3.2 concern the requirements elicitation process covering the elicitation from all relevant sources. Section 4.3.1 focuses on the risk-driven requirements elicitation from the PFMEA, while Section 4.3.2. focuses on elicitation of requirements from other design constraints, such as manufacturing specifications, interviews, as well as observation of the actual environment. As the products used in the test builds are excluded from the shipments to the end customer, the iterative life-cycle approach emphasizes delivering functional software first in order to minimize the manual work required in the process before covering the risk-based requirements. Also, the risk-based requirements are prone to evolving during the process testing, highlighting the importance of iterative development. Next, the section 4.3.3. covers the functional and non-functional requirements analysis. These requirements result from the iterative elicitation process and, as acknowledged in 2.3.2.2. may not have been developed to the final form. Further study and structure were needed to ensure the individual and set of requirements aligned with the desired characteristics identified from 2.3.2.2, and The Guide to the Software Engineering Body of Knowledge (SWEBOK v4.0). practises.

4.3.1 Acquiring Design Inputs from PFMEA

The PFMEA served as the main source of design inputs for the software system. Document analysis of the relevant design verification PFMEA, in addition to the manufacturing specification, was conducted to elicit and form the foundation for the risk-based requirements. These sources also form the basic understanding of the process in order to identify essential mitigation plans, operations needed for monitoring, as well as automated error-prevention functionalities needed for the system. In case new risks, changes to the risk assessment, or failure modes were identified during the test builds additional rows and changes were made to the PFMEA worksheet. Their causes and effects were analysed, and Risk Priority Numbers (RPNs) were assigned to Impact, Probability, as well as Detection. For clarification, Impact translates to Severity in this context. In addition to the comprehensive document analysis, an initial brainstorming meeting was organized to ensure initial mutual understanding of the risks. Storytelling was used to provide input from the operators, as well as experts, during the test builds throughout the development process.

The PFMEA worksheet contains process functions, failure modes, causes, and controls in a traceable manner similar to Figure 5. The controls that the software needs to address in terms of automated data collection, operator inputs, as well as workflow were constructed to the (RTM) Requirements Traceability Matrix, where each risk can be further prioritized and linked to the appropriate software requirement item. This made the handling of risk-based functional and non-functional requirements easier, in addition to ensuring that the traceability system, as well as the software, complies with the process reliability, mitigating adequate risks. The identified risks from the process are listed in Table 3.

ID	Failure mode	Cause	Local / System Effect	Imp	Prob	Det	RPN
1	QR-code not scanned into eDHR	Manual QR-code data error	Incorrect eDHR records and loss of traceability / No system impact	3	3	2	18
2	Incorrect test result input to eDHR	Manual data entry error	Incorrect eDHR records and process capability / No system impact	3	3	2	18
3	Incorrect recipe selected by operator to the eDHR	Manual data entry error	Incorrect eDHR records. / No system impact.	3	3	2	18

14	Duplicate product identifiers marked in the QR-code	Previously used product identifier marked in to laser on the same day	Multiple products scrapped - loss of traceability / No system impact	3	3	3	27
29	Ink not transferred through the opening in the screen mesh	Incorrect recipe used to print conductives . Ink drying on screen.	Poor definition of printed traces causing shorts / Electrical failure of product	4	3	2	24
40	Trace thickness is printed too thin	Incorrect recipe used to print conductives . Ink drying on screen.	Scrap – trace resistance exceeds specs / Electrical failure of product	4	3	2	24
50	Conductive inks bleeding between traces	Incorrect recipe used to print conductives . Ink drying on screen.	Scrap – shorting between traces / Electrical failure of product	4	3	2	24
83	Increased printing cycle times	Incorrect recipe used to print graphics	Loss of production hours / No system impact	4	3	2	24
115	Increased cycle times	Incorrect recipe used to print dielectric	Loss of production hours / No system impact	4	3	2	24
56	High conductive inks resistance	Incorrect (UV) oven profile used, conductive ink not cured	Scrap - incorrect resistances / Electrical failure of product	5	2	3	30
59	Electrical failure	Product processed through oven before reaching temperature	Lost production, electrical failure	5	2	1	10
60	Wrinkled or shrunk sheet	Incorrect (stabilization) profile used – oven too hot	Scrap – Lost production / Electrical failure of product	4	2	3	24
116	Dielectric or graphic inks not drying, adhering to substrate	Conductive or stabilization profile used, preventing curing	Scrap – wet ink removed from product, reducing insulation resistance or application instructions / Electrical failure of product or unclear application instructions.	5	2	3	30
118	UV cured inks not drying	Product processed through oven before reaching	Scrap – wet ink removed from product, reducing insulation resistance or	5	2	1	10

		acceptable UV parameters	application instructions / Electrical failure of product or unclear application instructions.				
--	--	--------------------------	---	--	--	--	--

Table 3: Risk Identification Process

A Pareto Chart was utilized to visualize the highest priority and help to determine their contribution to the overall risk. It provides traceability to the risks and helps to prioritize the risk mitigation with the highest RPNs.

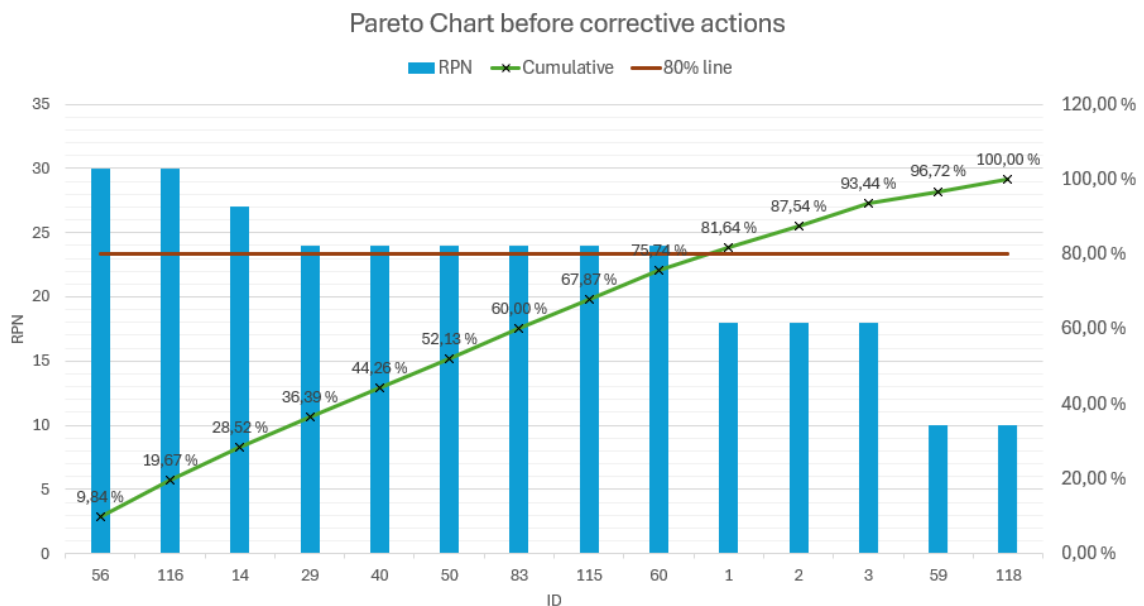


Figure 25: Pareto Chart Before Corrective Actions

From Figure 25, we can see which individual failure modes (IDs) cumulatively contribute to approximately the overall 80% of the overall process risks. These individual failure modes were prioritised in the design of the software, with appropriate mitigation strategies. In Figure 25, the 80% line is crossed in ID 1, referring to the items 56, 116, 14, 29, 40, 50, 83, 115, 60, and 1 contributing to the overall 80% of the total process risk.

4.3.2 Acquiring Design Inputs from Design Constraints

The design constraints in this system refer to the environmental, process-specific, technical, as well as organizational boundaries required for the software. This also includes requirements derived directly from Chapter 4.2 IoT-enabled system framework in which the software needs to be able to operate. Further, these requirements help to fulfil the risk-based requirements, enhancing the risk mitigation process.

The software requirements are affected by the direct and indirect marking techniques used for product and materials identification due to the geometry of these parts. As an example, the software shall be able to handle QR code read data and parse it according to the specified rules. The equipment used for the manufacturing process set limitations for the use of OPC UA to all equipment, as replacing or modifying existing equipment was not possible. As a result, the software needs to be able to handle Modbus TCP/IP communication protocol in addition to OPC UA. Finally, the software needs to support the proposed cloud-based eDHR infrastructure. This forms the mandatory organizational requirement for all process and product records throughout the manufacturing process. The software needs to adopt the sequential workflow logic defined in the Data Collection Plans (DCPs) incorporated into the eDHR structure. Traditional elicitation techniques, such as interviews, were used to elicit requirements imposed by the cloud-based framework.

Contextual techniques, as discussed in 2.3.2.1, were used to elicit requirements in the context of the user. Specifically, observation was used to analyse the end user in the manufacturing environment to elicit requirements that the process imposes. An ethnographic view was formed from interacting with the stakeholders to gain an understanding of the organizational environment. As a result of the process, a software logic driven by the environment, as well as previously discussed constraints, was proposed and visualized using a flowchart in Figure 26.

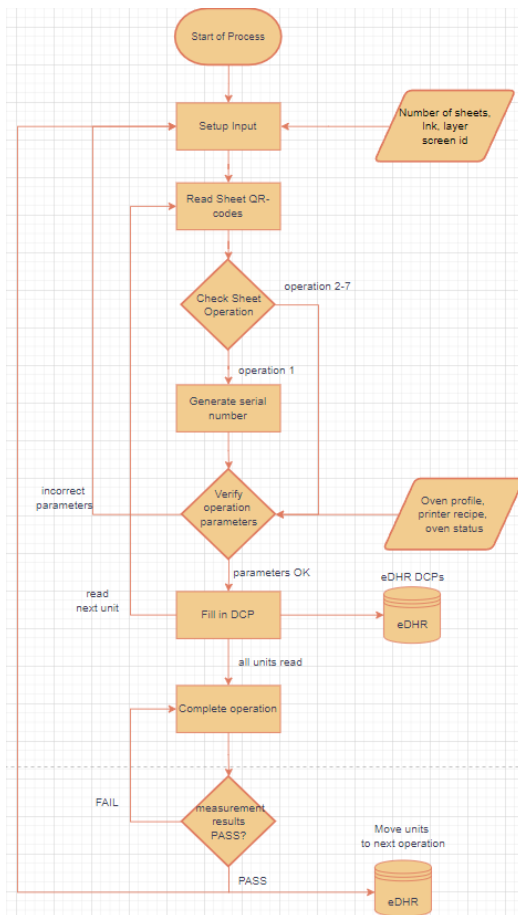


Figure 26: High-Level Proposed Software Logic

4.3.3 Development of Functional and Non-functional requirements from Design Inputs

The development of functional and non-functional requirements was carried out iteratively to deliver functional software to test builds early on, as the manufacturing process lacked a clear automated data collection framework from the test builds. This evolutionary requirement prototyping approach was utilized to form a small set of quality-driven requirements to create a prototype of the system, improving the software continuously throughout the test builds. In addition, this supported the adopted iterative life-cycle method, which is included in more detail in Figure 6. The iterative methods also support deriving and enhancing requirements and enhancing the test builds, supporting the previously discussed contextual techniques, as well as adding the opportunity to address risks iteratively, as discussed in 2.4.

Table 4 contains the specific iterations of the functional and non-functional requirements, as well as highlights the prototyping and iterative nature of the development.

Version:	Software functionality:	Development process
N/A	N/A - Manual collection at first test build	Initial eDHR structure with sequential rules and DCPs tested
1.0	Functionality-driven software from the initial set of requirements and automated eDHR DCP collection.	Initial set of quality-driven functional and non-functional requirements for eDHR and DCP adoption.
2.0	Safety-critical functions, automation, material identification, completeness.	Enhancements to the initial requirements, risk-driven requirements, equipment automation, and serialization rules.
3.0	Final bug fixes and performance tweaks.	Requirements validation and polishing
4.0	Formal validation process of the software.	SRS Requirements frozen.

Table 4: Software Iterations

4.3.4 SRS Content and Structure

The derived functional and non-functional requirements were documented in the Software Requirements Specification (SRS). The SRS outline was derived from ISO/IEC/IEEE 29148:2018(E), with modifications and exclusions made if certain constraints and sections were not considered to provide value in terms of the requirements. The following biography was adopted to the software requirements specification in Figure 27.

- 1 Introduction**
- 1.1 Purpose
- 1.2 Scope
- 1.3 Product Overview
- 1.3.1 Product Perspective
- 1.3.2 Hardware and Communication Interfaces
- 1.3.3 Product functions
- 1.3.4 User Characteristics and Classes
- 1.3.5 Limitations
- 1.4 Definitions
- 2 References**
- 2.1 Product Specific Reference Documents
- 2.2 General Reference Documents
- 3 Requirements**
- 3.1 Functional Requirements
- 3.2 Non-Functional Requirements
- 3.2.1 Performance Requirements
- 3.2.2 Usability Requirements
- 3.2.3 Interface Requirements
- 3.2.4 Logical Database Requirements
- 3.2.5 Design and Implementation Constraints
- 3.2.6 Software System Attributes
- 3.2.7 Compliance
- 3.3 Other Requirements
- 4 Appendices**
- 4.1 Assumption and dependencies
- 4.2 Acronyms and abbreviations

Figure 27: SRS Bibliography

The introduction chapter includes sections for purpose, scope, product overview, and definitions. The purpose contains a rationale for developing the software, in this case, the purpose of the software is to automate the data collection, mitigate manufacturing risks, and generate traceable records from the manufacturing process. The scope defines the process steps affected by the software, referring to the screen printing, oven curing, and thickness measurement phases. Product Overview sections contain a product perspective sub-chapter defining how the software fits into the larger system in terms of the manufacturing process. A block diagram was used to showcase the system as a part of the manufacturing process. Different interfaces were covered in the Hardware and Communications interfaces, using a visualization of the Figure 24 architecture diagram. Product functions referring to the major functions that the software will perform were carried out using natural language at a high level, while emphasizing the software logic proposed in Figure 26. User characteristics describe the intended groups of users of the software and their required functions of the software. As an example, the operator requires visual guidance of the software, and engineering requires access to the configuration file in case of changes in specific software parameters and troubleshooting. The limitations addressed in the scope of this system are the communication interfaces regarding the OPC UA and Modbus TCP/IP. These manufacturing equipment-specific frameworks limit the current system, as well as may impose limitations in case changes to the equipment are made. Definitions are intended to define the specification type. In this case, natural language specification is used, where mandatory functional requirements are indicated using “shall” and desired optional characteristics are expressed using “should”. The References section provides trace links to related documents, blueprints, as well as specifications related to SRS, and general references provide information on the related general guidelines followed.

The requirements section presents the specific functional and non-functional requirements of the software in a structured natural language. The functional requirements are uniquely identified by a naming format **[TST-LCNMT-010]**, as an example. The structured action-actor approach was used to provide structure to individual requirements. The actor in this context is often associated with the operator and the action, the action what function the software needs to execute. An example of the previous requirement is expressed as “When a sheet QR code without existing records is read to the software, a new serial number shall be generated to eDHR”. In addition to the previously mentioned tag, used for tracing a short description of the requirement is added to provide a description of the requirement in this case, “Generate eDHR serial number”. The individual requirements and the collection of the requirements are evaluated further in 5.2 based on predefined criteria.

The non-functional requirements section contains various other requirements regarding performance, usability, interfaces, logical databases, design constraints, software attributes, and compliance. Performance requirements contain the key requirements for operator interactions between the system, as well as desired high-level processing times in order to minimize the processing times per unit. The interface requirements shall follow the architectural design interfaces highlighted in Figure 21. Usability requirements covered the user interfaces required for the operator to provide visual guidance to the manufacturing parameters, as well as the software should only require the use of a barcode scanner outside the setup phase. The Logical database requirements form the specific data formats, which the software should be able to handle. This includes the direct part marking QR code format, indirect part marking material QR code format, screen identifier data, oven profile data from OPC UA, as well as printer recipe data from Modbus TCP/IP. The design and implementation constraints guide through the desired requirements for the system, such as eDHR used as the primary database and operating system requirements.

The system attributes section provides details and rationales on the desired attributes of the software. Availability covers the estimated uptime requirements for the operation of the software during production. Scalability estimates the capacity of production units that the software needs to be able to handle during production. As the software is used specifically at GE HealthCare facility in a controlled environment, with extensive internal security policies, as a result, no personnel outside GEHC is able to access the software, resulting in no further attributes in security and portability. Software will be supported by the internal engineering team and maintained by recording relevant installation packets in the documentation system. Software compliance with the industry standards is ensured by following a specific design, development, and release process. Appendices clarify the acronyms and abbreviations used in the specification.

5 Results

This Chapter presents the results of the thesis proposed software system, starting from the identified risks and their mitigation between different iterations of the software. Results from the SRS and traceability of the requirements are covered in 5.2. The system and the digital records produced by the system are evaluated in 5.3.

5.1 Risk Mitigation Results

The Risk Mitigation Results subsection evaluates the effectiveness of the proposed software system in mitigating the identified risks during different test build iterations of the software. Due to the tight schedule of the project, not all functionalities are implemented in the first iteration of the software labeled as version 1.0 in Table 4. As the test builds are not shipped to the end user, the risks and controls can be evaluated and enhanced for the next iteration of the software covered in versions 2.0-4.0. The RPNs of the identified risk are presented before and after the implementation of the software in both iterations.

5.1.1 Risk Mitigation Results from the Initial Software Version

The first iteration of the proposed software provides enough functionalities so that it can be tested in the actual manufacturing environment. The software provides an interactive interface that automatically excludes the possibility of the user selecting an incorrect input, resulting in erroneous or missing audit trails. However, automated communication functionality between the oven and the printer was not yet implemented in this version of the software, which impacted the overall risk mitigation effectiveness. Table 5 below showcases the RPNs after the first version of the software implementation.

Many of the risk mitigation actions were not yet implemented in the first version of the software, which emphasized delivering functional software to the test builds, not fully covering all risks. The affected risk items where mitigation occurred are represented in Table 5.

ID	Failure mode	Cause	Mitigation	Imp	Prob	Det	RPN *	Change-%
1	QR-code not scanned into eDHR	Manual QR-code data error	SW does not allow erroneous input	3	3	1	9	-50

3	Incorrect recipe selected by operator to the eDHR	Manual data entry error	SW guides to select the correct printer recipe	3	2	2	12	-33
14	Duplicate product identifiers marked in the QR-code	Previously used product identifier marked in to laser on the same day	SW detects the duplicate sheet numbers	3	3	2	18	-33
56	High conductive based inks resistance	Incorrect (UV) oven profile used, conductive ink not cured	SW Oven profile selection guidance	5	1	3	15	-50
60	Wrinkled or shrunk sheet	Incorrect (stabilization) profile used – oven too hot	SW Oven profile selection guidance	4	1	3	12	-50
116	Dielectric or graphic inks not drying, adhering to substrate	Conductive or stabilization profile used, preventing curing	SW Oven profile selection guidance	5	1	3	15	-50

Table 5: First Iteration of Risk Mitigation

5.1.2 Risk Mitigation Results from The Revised Software

The versions 2.0-4.0 of the software included enhancements to automation between the oven and the printer, further mitigating the possibility of operator erroneous inputs. The second iteration of the software evaluates the mitigations in contrast to the initial identified risks in Table 3 and presents the full risk mitigation table and results after proposed controls. The full risk mitigation table after the second iteration is presented in Table 6.

ID	Failure mode	Cause	Mitigation	Imp	Prob	Det	RPN*	Change -%
1	QR-code not scanned into eDHR	Manual QR-code data error	SW does not allow erroneous input. Display time between the last successful QR-code read.	3	3	1	9	-50
2	Incorrect test result input to eDHR	Manual data entry error	SW automatic transfer to eDHR	3	1	2	6	-67
3	Incorrect recipe selected by operator to the eDHR	Manual data entry error	SW automatic recipe confirmation	3	1	2	6	-67
14	Duplicate product identifiers marked in the QR-code	Previously used product number marked in to laser on the same day	SW detects the duplicate product identifiers	3	3	2	18	-33
29	Ink not transferred through the opening in the screen mesh	Incorrect recipe used to print conductive . Ink drying on screen.	SW detects the correct conductive recipe and does not allow to proceed	4	3	1	12	-50
40	Trace thickness is printed too thin	Incorrect recipe used to print conductive . Ink drying on screen.	SW detects the incorrect conductive recipe and does not allow to proceed.	4	3	1	12	-50
50	Conductive inks bleeding between traces	Incorrect recipe used to print conductive . Ink drying on screen.	SW detects the correct conductive recipe and does not allow to proceed.	4	3	1	12	-50
83	Increased printing cycle times	Incorrect recipe used to	SW detects the correct graphics recipe and does	4	3	1	12	-50

		print graphics	not allow to proceed.					
115	Increased cycle times	Incorrect recipe used to print dielectric	SW detects the correct dielectric recipe and does not allow to proceed.	4	3	1	12	-50
56	High conductive inks resistance	Incorrect (UV) oven profile used, silver not cured	SW automatically detects the incorrect oven profile for conductives	5	1	1	5	-83
59	Electrical failure	Product processed through oven before reaching temperature	SW automatically reads and checks the oven status during each unit read.	5	1	1	5	-50
60	Wrinkled or shrunk sheet	Incorrect (stabilization) profile used – oven too hot	SW automatically detects if stabilization is used for the wrong operation	4	1	1	4	-83
116	Dielectric or graphic inks not drying, adhering to substrate	Conductive or stabilization profile used, preventing curing	SW automatically detects if incorrect oven profile for dielectric and graphic is used	5	1	1	5	-83
118	UV cured inks not drying	Product processed through oven before reaching acceptable UV parameters	SW automatically reads and checks the oven status during each unit read.	5	1	1	5	-50

Table 6: Second Iteration of Risk Mitigation

The Pareto diagram shows the effectiveness of the risk mitigation results after the implementation of the final software iterations.

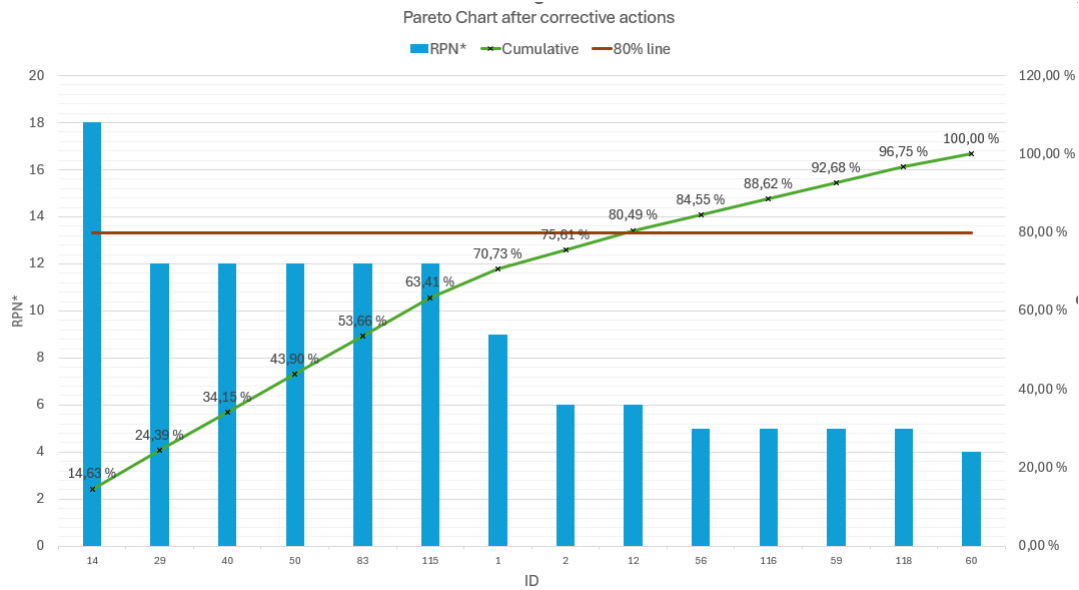


Figure 28: Pareto Chart After Corrective Actions

In Figure 28, after the risk mitigation process, the maximum RPN was reduced from 30 to 18, showcasing the effectiveness of the corrective actions in reducing overall risk priority. This reduction primarily improved the detection and occurrence probability through automation and validation controls. The cumulative threshold of 80% is now reached with fewer and lower RPN failure modes. The high-priority failure modes from the previous Figure 25 IDs 56, 116, 14, 29, 40, 50, 83, 115, 60, and 1 were mitigated successfully.

5.2 SRS Document and Requirements Evaluation

To ensure unambiguity, verifiability, and feasibility of the individual SRS requirements, as well as a complete and consistent SRS document, the following table with check boxes was used in the review of the requirements with the corresponding software engineer to verify that each requirement was unambiguous, verifiable, and feasible after the iterations of the software development process. The check boxes were verified in the review meeting organized by the author.

Requirements review				
Requirement	Unambiguous	Verifiable	Feasible	Notes:
[TST-LCNMT-001]	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	not verifiable or feasible in the present form
[TST-LCNMT-002]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	ambiguity in result format
[TST-LCNMT-003]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
[TST-LCNMT-004]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
[TST-LCNMT-005]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
[TST-LCNMT-006]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
[TST-LCNMT-007]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
[TST-LCNMT-008]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
[TST-LCNMT-009]	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	considered as a non-functional or constraint
[TST-LCNMT-010]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
[TST-LCNMT-011]	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	missing requirement
[TST-LCNMT-012]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Non-functional requirements	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	few not feasible system attributes
Specification table	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Overall SRS		Complete	Consistency	
		<input type="checkbox"/>	<input checked="" type="checkbox"/>	not complete based on individual requirements

Figure 29: SRS Review Meeting

Figure 29 shows the conflicts in requirements at the first review meeting. These conflicts were resolved by negotiating the differences between requirements in order to find consensus on the requirements. The general issues were related to the few additional requirements to enable more clear test case derivation, as well as few rephrased requirements due to ambiguity or verifiability. After the second iteration of requirements review, the check-sheet-based review met all the predefined criteria, and the requirements were frozen. The check boxes were verified in the review meeting organized by the author. It acted as the final check that the iteratively developed requirements were showcased and understood by the software engineer.

As the software requirements used in the SRS were elicited from various sources, RTM (Requirements Traceability Matrix) was established to provide traceability and structure to the development process. This was evaluated to be a valuable tool, consistently maintaining the traceability of the requirements until the final frozen requirements were reached, as well as mapping the requirements to appropriate test cases. The following worksheet in Figure 30, extended from Figure 2, illustrates the final traceability relationships between the SRS, the RTM, and the PFMEA, and test cases demonstrating how individual software requirements map to specific risk controls, process needs, and manufacturing constraints.

pFMEA reference				SRS reference		DQ Test Case
pFMEA ID:	Failure Mode	Potential Causes:	RPN Before Corrective actions	SRS ID:	SRS Requirement Description:	Test Case ID:
29	Ink not transferred through the opening in the screen mesh	Incorrect recipe used to print conductives . Ink drying on screen.	24	[TST-LCNMT-003]	Correct printer recipe shall be automatically read and checked, when a product QR-code is read into the software according to acceptable parameters in Table 2.	[LCNMT-DQ-001]
40	Trace thickness is printed too thin	Incorrect recipe used to print conductives . Ink drying on screen.	24			
50	Conductive inks bleeding between traces	Incorrect recipe used to print conductives . Ink drying on screen.	24			
83	Increased printing cycle times	Incorrect recipe used to print graphics	24			
115	Increased cycle times	Incorrect recipe used to print dielectric	24			

Figure 30: RTM Used For Requirements Traceability.

Using the Requirements Traceability Matrix (RTM) in Figure 30 ensured that each pFMEA ID, control action is mapped to the correct software requirement ID and test case ID validation. This resulted in a complete test coverage of the implemented software as each requirement was directly mapped to the pFMEA, as well as bidirectional traceability. In addition, the prioritization of high-risk failure modes was effective when using the RTM.

5.3 eDHR Records, Traceability, and System Performance Evaluation

This Chapter evaluates the eDHR records produced by the system and analyses the records produced by the system in terms of completeness and traceability. The effectiveness of the automated data collection was evaluated in terms of efficiency compared to the manual process, as well as how well the system was able to remove the manual work required for the production process.

5.3.1 Integrity and Traceability of the eDHR Records Audit Trail

The proposed system was successfully able to collect the required data for the traceability system framework, and the proposed software produced clear, traceable unit-level records from the manufacturing process in real-time. These records demonstrated the full end-to-end traceability, including materials, as emphasized in DPP fundamentals. Furthermore, the records can be accessed anytime by authorized personnel in case of recalls or complaints from the customer, supporting fast root cause analysis. These records were created for each product unit for every eDHR operation affected by the system.

To evaluate the traceability, queries to the database were executed to ensure the integrity of the audit trail as well as the completeness of the records. This evaluation confirmed that all process events and materials were reflected successfully in the product identifiers and executed in the proposed sequence. These records are visualized in Figures 31-37 and reflect the proposed sequential eDHR DCPs similar to Figure 19. These figures represent

results from a single unit during a test run. Queries to the eDHR were done after the test sequence, and the results demonstrated complete digital records of the product during and after the test run. Parameters were collected using m2m communication from the devices as well as direct and indirect part markings.

Figure 31: Operation 1 Digital Record

Line#	Prompt	Input	Result	User ID
10	Record screen ID	TV2003.000	ACCEPT	504009096 (Auto_Test Auto_Test)
20	Record printer recipe	DIELEC	ACCEPT	504009096 (Auto_Test Auto_Test)
30	Record Ink	123456	ACCEPT	504009096 (Auto_Test Auto_Test)
40	Record oven profile	UV Cure	ACCEPT	504009096 (Auto_Test Auto_Test)
50	Record ink expiration date	2026.09.09	ACCEPT	504009096 (Auto_Test Auto_Test)
60	Record ink Part Number	5966304	ACCEPT	504009096 (Auto_Test Auto_Test)

Figure 32: Operation 2 Digital Record

Line#	Prompt	Input	Result	User ID
10	Record screen ID	TV2004.000	ACCEPT	504009096 (Auto_Test Auto_Test)
20	Record printer recipe	SILVER	ACCEPT	504009096 (Auto_Test Auto_Test)
30	Record Ink	5D0991	ACCEPT	504009096 (Auto_Test Auto_Test)
40	Record oven profile	Silver Cure	ACCEPT	504009096 (Auto_Test Auto_Test)
50	Thickness test result (Pass/Fail)	PASS	ACCEPT	504009096 (Auto_Test Auto_Test)
60	Thickness test result YV	9.07	ACCEPT	504009096 (Auto_Test Auto_Test)
70	Thickness test result YO	9.09	ACCEPT	504009096 (Auto_Test Auto_Test)
80	Thickness test result AO	9.06	ACCEPT	504009096 (Auto_Test Auto_Test)
90	Thickness test result AV	9.00	ACCEPT	504009096 (Auto_Test Auto_Test)
100	Record ink expiration date	2026.02.18	ACCEPT	504009096 (Auto_Test Auto_Test)
110	Record ink Part Number	5873926	ACCEPT	504009096 (Auto_Test Auto_Test)

Figure 33: Operation 3 Digital Record

Line#	Prompt	Input	Result	User ID
10	Record screen ID	TV2005.000	ACCEPT	504009096 (Auto_Test Auto_Test)
20	Record printer recipe	SILVER	ACCEPT	504009096 (Auto_Test Auto_Test)
30	Record Ink	5H0561	ACCEPT	504009096 (Auto_Test Auto_Test)
40	Record oven profile	Silver Cure	ACCEPT	504009096 (Auto_Test Auto_Test)
50	Thickness test result (Pass/Fail)	PASS	ACCEPT	504009096 (Auto_Test Auto_Test)
60	Thickness test result YV	20.15	ACCEPT	504009096 (Auto_Test Auto_Test)
70	Thickness test result YO	20.18	ACCEPT	504009096 (Auto_Test Auto_Test)
80	Thickness test result AO	20.17	ACCEPT	504009096 (Auto_Test Auto_Test)
90	Thickness test result AV	20.17	ACCEPT	504009096 (Auto_Test Auto_Test)
100	Record ink expiration date	2026.02.18	ACCEPT	504009096 (Auto_Test Auto_Test)
110	Record ink Part Number	5873919	ACCEPT	504009096 (Auto_Test Auto_Test)

Figure 34: Operation 4 Digital Record

Line#	Prompt	Input	Result	User ID
10	Record screen ID	TV2006.000	ACCEPT	504009096 (Auto_Test Auto_Test)
20	Record printer recipe	DIELEC	ACCEPT	504009096 (Auto_Test Auto_Test)
30	Record Ink	250821	ACCEPT	504009096 (Auto_Test Auto_Test)
40	Record oven profile	UV Cure	ACCEPT	504009096 (Auto_Test Auto_Test)
50	Record ink expiration date	2027.08.20	ACCEPT	504009096 (Auto_Test Auto_Test)
60	Record ink Part Number	5873920	ACCEPT	504009096 (Auto_Test Auto_Test)

Figure 35: Operation 5 Digital Record

Line#	Prompt	Input	Result	User ID
10	Record screen ID	TV2006.000	ACCEPT	504009096 (Auto_Test Auto_Test)
20	Record printer recipe	DIELEC	ACCEPT	504009096 (Auto_Test Auto_Test)
30	Record ink	250821	ACCEPT	504009096 (Auto_Test Auto_Test)
40	Record oven profile	UV Cure	ACCEPT	504009096 (Auto_Test Auto_Test)
50	Thickness test result (Pass/Fail)	PASS	ACCEPT	504009096 (Auto_Test Auto_Test)
60	Thickness test result YV	38.47	ACCEPT	504009096 (Auto_Test Auto_Test)
70	Thickness test result YO	38.24	ACCEPT	504009096 (Auto_Test Auto_Test)
80	Thickness test result AO	38.32	ACCEPT	504009096 (Auto_Test Auto_Test)
90	Thickness test result AV	37.86	ACCEPT	504009096 (Auto_Test Auto_Test)
100	Record ink expiration date	2027.08.20	ACCEPT	504009096 (Auto_Test Auto_Test)
110	Record ink Part Number	5873920	ACCEPT	504009096 (Auto_Test Auto_Test)

Figure 36: Operation 6 Digital Record

Line#	Prompt	Input	Result	User ID
10	Record screen ID	TV2007.000	ACCEPT	504009096 (Auto_Test Auto_Test)
20	Record printer recipe	SILVER	ACCEPT	504009096 (Auto_Test Auto_Test)
30	Record ink	ZF30225	ACCEPT	504009096 (Auto_Test Auto_Test)
40	Record oven profile	Silver Cure	ACCEPT	504009096 (Auto_Test Auto_Test)
50	Record ink expiration date	2027.08.20	ACCEPT	504009096 (Auto_Test Auto_Test)
60	Record ink Part Number	5873921	ACCEPT	504009096 (Auto_Test Auto_Test)

Figure 37: Operation 7 Digital Record

Line#	Prompt	Input	Result	User ID
10	Record screen ID	TV2007.000	ACCEPT	504009096 (Auto_Test Auto_Test)
20	Record printer recipe	SILVER	ACCEPT	504009096 (Auto_Test Auto_Test)
30	Record ink	ZF30225	ACCEPT	504009096 (Auto_Test Auto_Test)
40	Record oven profile	Silver Cure	ACCEPT	504009096 (Auto_Test Auto_Test)
50	Thickness test result (Pass/Fail)	PASS	ACCEPT	504009096 (Auto_Test Auto_Test)
60	Thickness test result YV	24.33	ACCEPT	504009096 (Auto_Test Auto_Test)
70	Thickness test result YO	24.64	ACCEPT	504009096 (Auto_Test Auto_Test)
80	Thickness test result AO	24.48	ACCEPT	504009096 (Auto_Test Auto_Test)
90	Thickness test result AV	24.53	ACCEPT	504009096 (Auto_Test Auto_Test)
100	Record ink expiration date	2027.08.20	ACCEPT	504009096 (Auto_Test Auto_Test)
110	Record ink Part Number	5873921	ACCEPT	504009096 (Auto_Test Auto_Test)

These digital records collected from the manufacturing process satisfied the completeness, integrity of the audit trail, and unit-level traceability. All of the parameters were successfully and efficiently collected, satisfying the SRS requirements. These data records are traceable and available at any stage of the manufacturing and the product's life cycle, further satisfying the end-to-end traceability requirements.

5.3.2 System Performance Evaluation

The performance of the implemented system was evaluated by comparing the automated data-collection workflow to the previous manual process. The evaluation highlights the improvements in process efficiency and reduction in manual data collection workload on two occasions.

First, in the printing setup phase operator is required to record materials, machine parameters, and process information to the DHR collection form manually before starting a batch. The time required for the setup was recorded with the automated process, where the same parameters are captured

automatically. Next, the operator needs to process the units and manually collect the data into the DHR form. This per-unit processing time is compared to the automated software's performance on how fast it fills the required fields to the eDHR cloud following a barcode scan. In addition, operator interactions are measured in the manual collection compared to the automated system. This means that each manual action, such as data entry, parameter verifications, and barcode scans, is compared between the approaches to analyze the workload reduction results. Table 7 highlights the performance evaluation results by comparing the time required for manual data entry and automated data capture on these production steps.

Description	Manual Collection	Automated System
Setup Phase data entry time	80 seconds	27 seconds
Setup phase operator interactions	6-7 interactions, including parameter confirmations and manual inputs	Single QR code scan, single manual interaction
DCP Collection per unit time	30 seconds	7 seconds
Operator interactions per unit	6-7 interactions, including parameter confirmations and manual inputs	Single QR code scan

Table 7: System Performance Evaluation Table

The automated traceability system was able to reduce the DHR collection time during the setup phase from 80 seconds to 27 seconds. More specifically, this time comes from different interactions, confirmations, and manual collection that the operator needs to perform before starting the production run. During the automated production run, the operator needs to manually collect the data into the collection form. This processing time took approximately 30 seconds. The automated system filled the required information into the eDHR at approximately 7 seconds. This was acceptable as the operator can immediately continue working after scanning the product QR code, and the software continues to send the data to the cloud. In contrast to the manual process, the operator needs to be present when inputting the data manually. The workload in manual collection required 6-7 different interactions and confirmations, which were prone to operator errors and potentially led to erroneous data being exposed in the DHR. With the automated implementation, the operator was only required to interact with the software using barcode scanners and one interaction during the setup phase. This significantly reduced the workload in the process phase, as the operator does not need to consistently confirm and input different parameters into the DHR form.

6 Conclusions and Future Work

This thesis studied how a real-time traceability system can be designed to support medical device batch manufacturing. The work focused on enabling unit-level traceability in order to improve the efficiency of data collection, as well as mitigate process risks using a systematic approach. This approach utilized serialization, IoT-based data collection, and electronic Device History Records (eDHR) with cloud integration. A structured software requirements specification was created and evaluated for the proposed system.

At the start of the work, risk management was conducted using a Process Failure Modes and Effects Analysis (PFMEA) approach. From the PFMEA worksheet, manufacturing failure modes relating to the process scope were identified and acted as design inputs for the system. In addition to the risk-based requirements derivation, manufacturing process specifications, environmental constraints, organizational constraints, as well as observations were used as design inputs and further developed into functional and non-functional requirements for the software requirements specification (SRS). The software requirement specification was reviewed for completeness and consistency, and individual requirements for feasibility, ambiguity, and verifiability. The systematic design process was supported using a Requirements Traceability Matrix (RTM) to prioritize high-risk requirements, as well as maintain traceability links between document artefacts.

To address the research questions, the proposed system framework utilizes direct laser engraving, as well as indirect serialization using QR codes and Data Matrix for identification and traceability. An IoT-enabled system framework was established following a 3-layered architecture with Perception, Transmission, and Application Layers, taking advantage of the manufacturing equipment's machine-to-machine communication protocols OPC UA and Modbus TCP/IP. A tailored data collection plans and sequential logic was integrated into the cloud-based eDHR to support the large amount of data produced by the IoT system. In addition, the traceability system fundamentals supported the Digital Product Passport (DPP) principles for materials and essential manufacturing parameters collection, creating individual unit-specific digital records from the production phases. The completeness of these records was validated through test runs as shown in Figures 31-37. The system performance was evaluated by analysing the automated data collection system in comparison to the manual data collection process by comparing the operator workload and data collection efficiency. The system was able to reduce the operator workload during manufacturing to a single barcode scan and eliminate the additional confirmations required in the manual process. Additionally, the data entry times were significantly lowered using the automated traceability system, and the availability of the records enabled fast root cause analysis of the production process and the unit, as highlighted in Table 7. Also, the implementation of the system efficiently lowered the

RPNs of the high-risk failure modes from the process, as demonstrated in Figure 28.

Currently, the project will undergo a formal manufacturing validation process. In case there are new failure modes identified after the manufacturing validation, this system can be re-evaluated and extended to mitigate future issues by expanding the data collection parameters or with a software update. Also, RFID integration was considered at the system development process for the traceability of the printing screens, but was eventually excluded due to the implementation would have added unnecessary steps to the operation, and no high-risk failure modes were identified in screen identification. The scalability of this system was considered for another facility sharing similarities in the manufacturing process, however the current system and software configuration are highly specific to the manufacturing process at Helsinki and would require configuration and software changes. The modularity constraints could be accessed in the future to enable more seamless adoption of the system. The DPP adoption in the current process supports the traceability to internal teams. At the time of writing the thesis, no specific implementation guidelines or standards were introduced by the European Union for implementing the DPPs to medical devices. This system already shares many fundamentals of the DPPs and could be modified to support wider integration by modifying the existing system. Real-time analytics and performance tuning could be of interest in the future. This can be facilitated by expanding the data collection parameters and implementing a modification to the software.

References

- [1] A. P. Joseph Neelamkavil, Michael Kernahan, "Traceability in Medical Devices Design & Manufacturing," in *Proceedings of the Canadian Engineering Education Association (CEEA)*, 15.08.2011 2005, doi: <https://doi.org/10.24908/pceea.voio.3968>. [Online]. Available: <https://ojs.library.queensu.ca/index.php/PCEEA/article/view/3968>
- [2] K. K. Aleksei Snatkina, Jüri Majaka, Tanel Aruvälia, Tanel Eiskopb, "Real time production monitoring system in SME," *Estonian Journal of Engineering* 19.1, pp. 62–75, 2013, doi: 10.3176/eng.2013.1.06.
- [3] X. X. Reuben Schuitemaker, "Product traceability in manufacturing: A technical review," presented at the 53rd CIRP Conference on Manufacturing Systems, 2020.
- [4] *ISO 9000:2015(en) Quality management systems – Fundamentals and vocabulary*, I. O. f. S. (ISO), 2015. [Online]. Available: <https://www.iso.org/obp/ui/#iso:std:iso:9000:ed-4:v1:en:term:3.2.1>
- [5] J. C.-H. Orlena Gotel, Jane Huffman Hayes, Andrea Zisman, Alexander Egyed, Paul Grünbacher, Alex Dekhtyar, Giulio Antonioli, Jonathan Maletic and Patrick Mäder, "Traceability fundamentals," in *Software and Systems Traceability*, 2012, pp. 3-22.
- [6] A. M. D. Duarte, *TraceBok - Guide of Software Traceability Body of Knowledge*. 2015.
- [7] W. M. N. W. Z. Abdulkadir Ahmad Madaki, "A Review on Tools and Techniques for Visualizing Software Requirement Traceability," presented at the Proceedings of the 11th International Conference on Robotics, Vision, Signal Processing and Power Applications, LNEE 829, 2022.
- [8] M. D. Mamta Madan, Anisha Tandon, "Need and Usage of Traceability Matrix for Managing Requirements," *International Journal of Engineering Research*, vol. 5, no. 8, pp. 666-668, 2016, doi: 10.17950/ijer/v5s8/805
- [9] T. U. K. Kamalabalan, G. Thiyagalingam, D. B. Wijesinghe, I. Perera, D. Meedeniya, D. Balasubramaniam "Tool Support for Traceability of Software Artefacts " presented at the Moratuwa Engineering Research Conference (MERCon), 2015.
- [10] S. Patel, "Statistical Approaches to Post-Market Surveillance and Device Safety in Medical Device Industries," *International Journal of Engineering Sciences & Research Technology*, vol. 8, no. 1, pp. 308-321, 2019, doi: 10.5281/zenodo.14598699.
- [11] M. B. Gilbert Regan, Fergal Mc Caffery, Kevin McDaid, Derek Flood, "A Traceability Process Assessment Model for the Medical Device Domain " in *Systems, Software and Services Process Improvement: 21st European Conference, EuroSPI 2014, Luxembourg, June 25-27, 2014. Proceedings 21*, Springer Berlin Heidelberg, 2014, pp. 206-216.

- [Online]. Available: <https://hdl.handle.net/10344/4389>. [Online]. Available: <https://hdl.handle.net/10344/4389>
- [12] Timiri S. Prakash Srinivasan, Pugazhenthana Thangaraju, Nandakumar Palanil, Thamizharasan Sampath, *Medical Device Guidelines and Regulations Handbook*: Springer Nature Switzerland AG, 2022. Accessed on: 10.07.2025.
- [13] R. P. M. Saveen A. Abeyratne, "Blockchain ready manufacturing supply chain using distributed ledger," *International Journal of Research in Engineering and Technology*, vol. 05, no. 09, pp. 1-10, 2016.
- [14] M. N. I. Iyolita Islam, "A blockchain based medicine production and distribution framework to prevent medicine counterfeit," *Journal of King Saud University - Computer and Information Sciences*, vol. 36, no. 1, 2024, doi: <https://doi.org/10.1016/j.jksuci.2023.101851>.
- [15] S. C. S. Sandeep Kumar Panda, "Drug traceability and transparency in medical supply chain using blockchain for easing the process and creating trust between stakeholders and consumers," *Personal and Ubiquitous Computing*, vol. 28, pp. 75-91, 2021, doi: <https://doi.org/10.1007/s00779-021-01588-3>.
- [16] X. L. Xiaoling Xia, Wenbin Dong and Zhi He, "Design of traceability system for medical devices based on blockchain " *Journal of Physics: Conference Series*, vol. 1314, 2019, doi: 10.1088/1742-6596/1314/1/012067.
- [17] G. M. Foivos Psarommatis, "Digital Product Passport: A Pathway to Circularity and Sustainability in Modern Manufacturing," *Sustainability*, vol. 16, no. 1, p. 396, 2024, doi: <https://doi.org/10.3390/su16010396>.
- [18] A. M. Hendro Wicaksonoa, Atit Bashyala, Tamas Feketea, "Digital Product Passport (DPP) technological advancement and adoption framework: A systematic literature review," *6th International Conference on Industry 4.0 and Smart Manufacturing*, vol. 253, pp. 2980-2989, 2025, doi: <https://doi.org/10.1016/j.procs.2025.02.022>.
- [19] S. M. P. Keyur K Patel, "Internet of Things-IOT: Definition, Characteristics, Architecture, Enabling Technologies, Application & Future Challenges " *International journal of engineering science and computing*, vol. 6, no. 5, 2016, doi: 10.4010/2016.1482.
- [20] M. Q. B.B. Gupta, "An overview of Internet of Things (IoT): Architectural aspects, challenges, and protocols.," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 21, 2020, doi: 10.1002/cpe.4946.
- [21] F. A. Anam Amjad, Muhammad Waseem Anwar, Wasi Haider Butt, "A Systematic Review on the Data Interoperability of Application Layer Protocols in Industrial IoT," *IEEE Access*, vol. 9, pp. 96528-96545, 2021, doi: 10.1109/ACCESS.2021.3094763.
- [22] K. N. Mageshkumar G, Tamilselvan K S, Suthagar S, Sharmila A. , "Design Of Industrial Data Monitoring Device Using Iot Through

- MODBUS Protocol " *International Journal of Scientific & Technology Research*, vol. 9, no. 1, 2020.
- [23] S. V. Alberto Morato, Federico Tramarin, Angelo Cenedese, "Assessment of Different OPC UA Implementations for Industrial IoT-based Measurement Applications," *IEEE Transactions on Instrumentation and Measurement*, vol. 70, pp. 1-11, 2020, doi: 10.1109/TIM.2020.3043116.
- [24] H. Y. T. Elton Kee Sheng Wong, Abdulwahab Funsho Atanda "Enhancing Supply Chain Traceability through Blockchain and IoT Integration: A Comprehensive Review " *Green Intelligent Systems and Applications*, vol. 4, no. 1, pp. 11-28, 2024, doi: <https://doi.org/10.53623/gisa.v4i1.355>.
- [25] A. J. Smith, "Framework for Establishing Asset Visibility and Traceability of Medical Devices," PhD, Department of Civil and Environmental Engineering, Massachusetts Institute of Technology, Doctoral dissertation, Massachusetts Institute of Technology, 2020.
- [26] M. D. C. G. (MDCG). "MDCG 2022-7 – Questions and Answers on the Unique Device Identification system under Regulation (EU) 2017/745 and Regulation (EU) 2017/746." European Commission. https://health.ec.europa.eu/document/download/b5429d14-25a9-4cfc-b059-355388f03e05_en?filename=mdcg_2022-7_en.pdf (accessed 10.02.2026, 2026).
- [27] U. S. F. a. D. Administration. "UDI Basics." U.S. Food and Drug Administration. <https://www.fda.gov/medical-devices/unique-device-identification-system-udi-system/udi-basics> (accessed 10.02., 2026).
- [28] A. A. T. Bożena Zwolińska, Norbert Chamier-Gliszczyński, Mariusz Kostrzewski, "Personalization of the MES System to the Needs of Highly Variable Production," *Sensors*, vol. 20, no. 22, 2020, doi: <https://doi.org/10.3390/s20226484>.
- [29] J. S. S. Soujanya Mantravadi, Charles Møller, "Application of MES/MOM for Industry 4.0 supply chains: A cross-case analysis," *Computers in industry*, vol. 148, 2023, doi: <https://doi.org/10.1016/j.compind.2023.103907>.
- [30] C. M. Soujanya Mantravadia, "An Overview of Next-generation Manufacturing Execution Systems: How important is MES for Industry 4.0?," *Procedia Manufacturing*, vol. 30, pp. 588-595, 2019, doi: <https://doi.org/10.1016/j.promfg.2019.02.083>.
- [31] S. N. Tushar Khinvasara, Nikolaos Tzenios, "Risk Management in Medical Device Industry," *Journal of Engineering Research and Reports*, vol. 25, no. 8, pp. 130-140, 2023, doi: 10.9734/JERR/2023/v25i8965.
- [32] D. P. Soniya Kundnani, Dhananjay Meshram, "Preliminary Hazard Analysis (PHA): Great approach to risk analysis in pharmaceutical industry," *International Journal of Frontiers in Science and Technology Research*, vol. 03(02), 2022, doi: <https://doi.org/10.53294/ijfstr.2022.3.2.0055>

- [33] S. S. Kapil Dev Sharma, "Failure Mode and Effect Analysis (FMEA) Implementation: A Literature Review," *Journal of Advanced Research in Aeronautics and Space Science*, vol. 5, no. 1&2, pp. 1-17, 2018.
- [34] D. C. G Cristea, "A comparative critical study between FMEA and FTA risk analysis methods," presented at the IOP Conf. Series: Materials Science and Engineering, 2017.
- [35] F. Z. A. S. Zuhair El attaoui, Youssef el Khatori, "Risk management for improving water quality: Application of the HACCP method," presented at the E3S Web of Conferences, 2023.
- [36] N. G. L. John M. Rising, "Systems-Theoretic Process Analysis of space launch vehicles," *Journal of Space Safety Engineering*, vol. 5, no. 3-4, pp. 153-183, 2018, doi: <http://dx.doi.org/10.1016/j.jsse.2018.06.004>.
- [37] S. D. Kilari, "Advanced Strategies for Automating PFMEA in Manufacturing Enhancing Efficiency and Risk Mitigation through Digital Integration," doi: <http://dx.doi.org/10.2139/ssrn.5234342>.
- [38] H.-C. Liu, *FMEA Using Uncertainty Theories and MCDM Methods*, Singapore: Springer Nature, 2016, pp. 13-27. Accessed on: 28.08.2025.
- [39] S. R. Soltanali Hamzeh, "Smart Failure Mode and Effects Analysis (FMEA) for the Safety-Critical Systems in the Context of Industry 4.0," presented at the Advances in reliability, failure and risk analysis, Singapore, 2023.
- [40] S. H. Benjamin Cabanes, Pascal Le Masson, Benoit Weil, "Improving reliability engineering in product development based on design theory: the case of FMEA in the semiconductor industry," *Research In Engineering Design*, pp. 309-329, 2021, doi: <https://doi.org/10.1007/s00163-021-00360-1>
- [41] S. Y. Johnny Marques, "An Analysis of Software Requirements Specification Characteristics in Regulated Environments," *International Journal of Software Engineering & Applications (IJSEA)*, vol. 10, 6, 2019, doi: 10.5121/ijsea.2019.10601.
- [42] J. I. O. Hironori Washizaki, *Guide to the Software Engineering Body of Knowledge v4.0*, S. T. Hironori Washizaki, ed.: IEEE Computer Society, 2024, pp. 1-23. Accessed on: 04.08.2025.
- [43] M. A. Masooma Yousuf, "Comparison of Various Requirements Elicitation Techniques " *International Journal of Computer Applications (0975 – 8887)*, vol. 116, no. 4, 2015.
- [44] S. S. R. Saurabh Tiwari, Atul Gupta, "Selecting Requirement Elicitation Techniques for Software Projects," presented at the CSI 6th International Conference on Software Engineering, CONSEG 2012, 2012.
- [45] J. G. Chetan Arora, Mohamed Abdelrazek, *Advancing Requirements Engineering through Generative AI: Assessing the Role of LLMs*, A. Nguyen-Duc, Abrahamsson, P., Khomh, F, ed.: Springer, Cham, 2024, pp. 129-148. Accessed on: 30.08.2025.

- [46] I. Sommerville, "Software Engineering, 10th edition," 10 ed.: Pearson Education, 2016, ch. 4.
- [47] M. I. Hafiz Anas Bilal, Qandeel Tariq, Muhammad Hummayun, "Requirements Validation Techniques: An Empirical Study," *International Journal of Computer Applications (0975 – 8887)*, vol. 148 No. 14, 2016, doi: 10.5120/ijca2016910911.
- [48] S. S. Massila Kamalrudin, "A Review on Software Requirements Validation and Consistency Management," *International Journal of Software Engineering and Its Applications*, 2015, doi: 10.14257/ijseia.2015.9.10.05.
- [49] M. K. B. Issa Atoum, Izzat Alsmadi, Ahmed Ali Otoom, Taha Alhersh, Jafar Ababneh, Jameel Almalki, Saeed Masoud Alshahrani, "Challenges of Software Requirements Quality Assurance and Validation: A Systematic Literature Review," *IEEE Access*, vol. 9, pp. 137613-137634, 2021, doi: 10.1109/ACCESS.2021.3117989.
- [50] M. W. Mwadulo, "Suitability of Agile Methods for Safety-Critical Systems Development: A Survey of Literature " *International Journal of Computer Applications Technology and Research* vol. 5, no. 7, pp. 465-471, 2016.
- [51] L. Tordrup Heeager, & Nielsen, P. A., "A conceptual model of agile software development in a safety-critical context: A systematic literature review," *Information and Software Technology*, no. 103, pp. 22-39, 2018, doi: 10.1016/j.infsof.2018.06.004.
- [52] S. S, "A Study of Software Development Life Cycle Process Models," presented at the National Conference on Reinventing Opportunities in Management, IT, and Social Sciences, 2017.
- [53] M. I. Hossain, "Software Development Life Cycle (SDLC) Methodologies for Information Systems Project Management," *International Journal for Multidisciplinary Research (IJFMR)*, vol. 5, no. 5, 2023.
- [54] P. D. Marco Kuhrmann, Jürgen Münch, Paolo Tell, Vahid Garousi, Michael Felderer, Kitija Trektere, Fergal McCaffery, Oliver Linssen, Eckhart Hanser, Christian R. Prause, "Hybrid software and system development in practice: Waterfall, scrum, and beyond," presented at the Proceedings of the 2017 International Conference on Software and System Process, 2017.
- [55] O. C. Martin McHugh, Fergal McCaffery, "An Agile V-Model for Medical Device Software Development to Overcome the Challenges with Plan Driven SDLCs " presented at the The Software Engineering in Healthcare (SEHC) Workshop at the 35th International Conference on Software Engineering (ICSE). San Francisco, California, USA, 2013.
- [56] F. M. C. Martin McHugh, Garret Coady, "An Agile Implementation within a Medical Device Software Organisation " presented at the The 14th International SPICE Conference Process Improvement and Capability Determination, 2014.

- [57] M. U. A. Aliya A. Khan, Wasi H. Butt, Mehreen Sirshar, "An Enhanced Agile V-Model: Conformance to regulatory bodies and experiences from model's adoption to medical device development.," *Heliyon* 10.6, Research Article 2024, doi: <https://doi.org/10.1016/j.heliyon.2024.e26928>.
- [58] Salvatore B. Mario Barbareschi, Riccardo Carbone, Valentina Casola, "Scrum for safety: an agile methodology for safety-critical software systems," *Software Quality Journal*, vol. 30, no. 4, pp. 1067-1088, 2022, doi: <https://doi.org/10.1007/s11219-022-09593-2>.