



CHALMERS
UNIVERSITY OF TECHNOLOGY



UNIVERSITY OF GOTHENBURG

Interactive Simulation Systems for Industrial Equipment

Supporting System Verification Through Interactive
Information Visualization

Master's thesis in Interaction Design & Technologies

JAKOB PERSSON
KEVIN TAN QUACH

MASTER'S THESIS 2026

Interactive Simulation Systems for Industrial Equipment

Supporting System Verification Through Interactive
Information Visualization

JAKOB PERSSON

KEVIN TAN QUACH



UNIVERSITY OF
GOTHENBURG



CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2026

Interactive Simulation Systems for Industrial Equipment
Supporting System Verification Through Interactive
Information Visualization
JAKOB PERSSON
KEVIN TAN QUACH

© JAKOB PERSSON, KEVIN TAN QUACH, 2026.

Supervisor: Staffan Björk, Interaction Design and Software Engineering
Advisor: Yuxuan Sun, Emerson
Examiner: Thommy Eriksson, Interaction Design and Software Engineering

Master's Thesis 2026
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in L^AT_EX
Gothenburg, Sweden 2026

Interactive Simulation Systems for Industrial Equipment
Supporting System Verification Through Interactive
Information Visualization
JAKOB PERSSON
KEVIN TAN QUACH
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Industrial process systems rely on precise level measurements to run safely and profitably. To verify that these measuring instruments work correctly under realistic conditions, engineers use simulation setups like the Interactive Plant Environment (IPE). However, exploring the massive amounts of live data generated during these tests is difficult without specialized tools. The primary purpose of this project is to answer how interactive simulation systems should be designed to support system verification. To investigate this, a prototype interface was developed to allow test engineers to easily run automated scripts, control the system manually, and analyze real-time data.

Using a Research through Design (RtD) approach and the Double Diamond process, the project started with semi-structured interviews to understand the daily workflows and frustrations of expert test engineers. Insights from this pre-study indicated that standard usability checklists are insufficient for live industrial environments. By combining these empirical findings with foundational Human-Computer Interaction (HCI) principles and Information Visualization (InfoVis) theory, the project introduces a set of custom design guidelines: spatial fidelity and dynamic state mapping, flexible test control, stable views for live data, clear error handling, proactive input validation and defensive safety nets, and data filtering.

Based on these guidelines, a functional web application prototype was built featuring visual tank models, live data plotting, and seamless switching between automated and manual execution. A heuristic evaluation of the prototype suggested that the interface supports expert test engineers by reducing cognitive load through visual metaphors and maintaining stable views during live data inspection. While the prototype managed complex testing scenarios, it also exposed vulnerabilities regarding conflicting commands and unguided error recovery, which directly resulted in refining and expanding the guidelines. Ultimately, this project proposes a set of domain-specific guidelines that can assist designers and developers in building interactive simulation and verification systems for similar industrial contexts.

Keywords: Information Visualization, Interaction Design, System Verification, Interactive Simulation, Test Automation.

Acknowledgements

We would like to express our sincere gratitude to everyone who supported us during the course of this project.

A special thank you goes to our advisor at Emerson, Yuxuan Sun, for making this project happen. We deeply appreciate your constant support and for always checking in on us, whether it was about the project or just to see how things were going.

We would also like to extend our deepest appreciation to our academic supervisor, Staffan Björk, for constantly helping us, providing valuable guidance, and for the many nice chats throughout the entire process.

Jakob Persson & Kevin Tan Quach, Gothenburg, 2026-06-17

Contents

List of Figures	xiii
1 Introduction	1
1.1 Research Question	1
1.2 Delimitations	2
1.3 Ethical Considerations	2
2 Background	5
2.1 Process Industries	5
2.1.1 Physical Systems in the Process Industry	5
2.1.2 Tank Gauging and Level Measurement	5
2.1.3 Distributed Control Systems	6
2.2 Emerson (Rosemount Tank Radar AB)	6
2.2.1 Interactive Plant Environment at Emerson	7
2.3 Related and Existing Work	7
2.3.1 Related Work	8
2.3.2 Existing Work	8
3 Theory	11
3.1 User Experience	11
3.1.1 Interaction Design	11
3.1.2 Usability	11
3.2 Accessibility Standards and Guidelines	12
3.3 Information Visualization	12
3.3.1 Methods of InfoVis	13
3.4 Scientific Visualization	14
3.4.1 Methods of SciVis	15
3.5 Wicked Problems	15
3.6 Personas	16
3.7 The 5 user guideline	16
4 Methodology	19
4.1 Research Through Design (RtD)	19
4.2 Double Diamond	19
4.2.1 Four phases of the Double Diamond	20
4.2.1.1 Discover	20

4.2.1.2	Define	22
4.2.1.3	Develop	25
4.2.1.4	Deliver	26
4.2.2	Evaluation Framework: Heuristics and Principles	26
5	Process	29
5.1	Project Planning and Execution	29
5.1.1	Project Phases	30
5.1.2	Scrum Framework	31
5.1.2.1	Sprints	33
5.1.2.2	Evolution of the Working Process	35
5.1.3	Design Process	35
5.2	Pre-study	35
5.2.1	Interviews	36
5.2.1.1	Participant Selection	36
5.2.1.2	Interview Procedure	37
5.2.2	Thematic Analysis	37
5.2.3	Themes from Interviews and Thematic analysis	38
5.2.4	Synthesizing the Design Guidelines	39
5.2.5	Initial Design Guidelines	40
5.3	Ideation and Concept Generation	41
5.3.1	Sketching	42
5.3.2	Voting Method	42
5.3.2.1	Voting results	42
5.3.2.2	Key strengths of the selected concept	44
5.3.2.3	Feedback from blue stickers	44
5.3.2.4	Rejected concepts	45
5.3.2.5	Outcome	45
5.4	Requirements Gathering	45
5.5	Implementation	46
5.5.1	Fading Scrum	46
5.5.2	Architectural and Design Impact of the Proposed Guidelines	46
5.5.2.1	Presenting the System State	47
5.5.2.2	Enabling Flexible Control	47
5.5.2.3	Temporal Data Synchronization and Plot Stability	47
5.5.2.4	Logging, Persistence, and Error Handling	47
5.6	Evaluation	48
5.7	Refined Design Guidelines	49
6	Results	53
6.1	Viper	53
6.1.1	System Overview	53
6.1.2	Data View	54
6.1.3	Test Tools	55
6.1.3.1	Scripts	55
6.1.3.2	Manual Mode	56
6.1.4	Backend	57

6.1.4.1	Overview	57
6.1.4.2	Entry Point and Configuration	58
6.1.4.3	API Layer	58
6.1.4.4	Real-Time Communication: ConnectionManager	58
6.1.4.5	Telemetry Streaming	59
6.1.4.6	Service Layer	59
6.1.4.7	Script Management	59
6.1.4.8	Design Decisions and Trade-offs	60
6.2	Proposed Guidelines	61
7	Discussion	65
7.1	Supporting System Verification Through Visualization	65
7.1.1	Interpreting the Evaluation Findings	66
7.2	Context of Use and Design Trade-offs	66
7.3	Limitations of the evaluation	67
7.4	Reflections over the Process	67
7.4.1	Evolution of the Workflow	67
7.4.2	Executing the Design Process	68
7.4.3	Technical Implementation and the Impact of Guidelines	68
7.4.4	Areas of improvement	69
7.4.5	Recommendations for Future Projects	69
7.5	Validity and Generalizability	70
7.5.1	Methodological Validity and Evaluation Bias	70
7.5.2	Generalizability of the Guidelines	71
7.6	Ethical and Social Considerations	71
7.7	Future work	72
8	Conclusion	73
	Bibliography	75
A	Appendix 1	I
A.1	Interview Guiding Questions	I
A.2	How Might We Questions	I

List of Figures

2.1	Illustration of a tank gauging setup.	6
2.2	Summary of Rosemount Tank Radar	7
3.1	Expected proportion of usability problems found as a function of the number of test users.	17
4.1	The Double Diamond by the Design Council [38].	20
4.2	Comparison of interview types based on question structure and flexi- bility [45].	21
4.3	The six phases of Thematic Analysis[47].	23
4.4	A basic three panel storyboard illustrating a user scenario.	24
5.1	Gantt chart - planned process for project	29
5.2	Gantt chart - actual process for project	30
5.3	Sprint plan	31
5.4	Example of a Sprint Retrospective board showing the sprint timeline and the team's voted notes.	33
5.5	Grouping of Postit notes during the thematic analysis, showing how key statements from the interviews were clustered into early theme categories.	38
5.6	Overview of all sketched concept ideas laid out after the sketching session.	42
5.7	Overview of all sketched concept ideas laid out after the voting session and before the voting.	43
5.8	Concept ideas that received green stickers during the voting.	43
5.9	Concept ideas that received blue stickers during the voting.	45
6.1	A screenshot of system overview	54
6.2	A screenshot of Data View	55
6.3	A screenshot of the part to handle Scripts	56
6.4	A screenshot of the part to handle Manual mode	57
6.5	Example of spatial fidelity	61
6.6	Example of flexible control modes	62
6.7	Example of paused live data view	63
6.8	Example of distinction between types of system messages	63
6.9	Data filtering	63

1

Introduction

Industrial process systems, such as those found in energy generation, chemical production, and water treatment are increasingly complex, software-driven, and safety-critical[1][2]. The measurement instrumentation is expected to operate and produce reliable measurements in varying conditions and circumstances[2]. With safety and profitability of processes relying on precise and reliable level measurements[3], manufacturers have strong incentives to ensure that their products and systems consistently meet their performance requirements. In an effort to meet and exceed these demands, realistic and representative tests are needed to better understand how the product behaves under field conditions, compared to lab conditions. A dedicated testing facility that bridges this gap has been constructed, but does not yet support test execution and the subsequent analysis of large volumes of data collected during testing.

An effective interface is an important component of the testing facility, users must be able to configure test scenarios, execute them, and explore collected data when the test is completed. Given the scale and complexity of modern industrial systems, the resulting datasets are often high-dimensional, time-dependent and heterogenous, making traditional analysis methods insufficient. Without appropriate tools, identifying patterns, anomalies or causal relationships in the data risk becoming difficult and time-consuming.

Information Visualization and Visual Analytics deal with taking large amounts of data and presenting it in an intuitive and understandable way[4]. As Ware[4] puts it, visual display is the method that transmits the most information from the computer to the human. Interactive visualization and visual analytics tools enable users to drill down into specific events and compare different test runs. Integrating robust visualization techniques into the testing interface supports more informed decision-making, reduces analysis time and contributes to the development of safer and more reliable industrial process systems.

1.1 Research Question

This project seeks to find an answer to *What should be considered when designing interactive simulation systems for industrial equipment to support system verification through interactive information visualization?*

Additionally, this project has a secondary goal of designing and implementing a

graphical user interface (GUI) that enables automation of test execution, manual control and result analysis. The GUI should also reflect the system state in real-time.

1.2 Delimitations

The scope of this project is strictly focused on designing and developing software that allows system verification engineers to run test scripts and analyse data. Because the core focus of this thesis is on interaction design and data visualization, several underlying technical areas are intentionally out of scope:

Hardware Setup: While the software naturally depends on physical equipment in the test environment, setting up, configuring, or maintaining that hardware is completely out of scope.

Data Logging and Collection: Although the software includes local logging for script execution, the collection of raw sensor data is managed by separate, pre-existing software. The interface acts as a bridge to these established systems rather than replacing them.

Data Storage: The company already has solutions in place for long-term data storage and database management. The project simply relies on what is already there instead of building any new architecture.

1.3 Ethical Considerations

This project engages test engineers and related roles involved in system verification. It is crucial to emphasize that while a software prototype is developed during this process, the prototype is merely an exploratory tool used within a Research through Design (RtD) framework. The overarching academic purpose of this project is to answer the central research question by formulating a set of design guidelines. Consequently, the ethical considerations of this project are not evaluated from the perspective of releasing a commercial software product. Instead, they are strictly framed around research ethics: ensuring that the process of generating these guidelines respects the privacy, daily workflows, and expertise of the participating test engineers.

Because the research is explicitly centered on extracting knowledge to answer the research question, it naturally includes these domain experts while excluding individuals outside this context, as their input would not directly inform the resulting guidelines. Furthermore, while the exploratory software prototype is tailored strictly for internal use within the partner company and is not intended for commercial distribution, the resulting design guidelines are intended to be shared academically. By generalizing the findings, these guidelines can assist designers and developers in similar industrial contexts without exposing proprietary company data. Ultimately, this approach ensures the project contributes to the broader academic community while protecting the operational security of the partner company, meaning the project does not pose direct ethical risks to the general public.

Since the work involves collaboration between a university and an industry partner, data handling is an important ethical consideration. Interviews may include information about workflows or internal practices, so all data is stored securely and only used for research purposes. To ensure participant comfort and confidentiality, no audio recordings were made during the interviews. Instead, data was collected through detailed written notes. These digital notes were stored securely within a private, access-restricted Overleaf repository used for compiling the thesis. Access to the raw data was strictly limited to the two primary researchers. Furthermore, to guarantee long-term data security, all raw interview data will be retained only for the duration of the examination period and will be permanently deleted from the repository upon the final grading and publication of this thesis.

The project does not make use of AI tools for data collection, analysis, or interpretation. All insights and conclusions are produced manually to ensure transparency and maintain full control over how the data is understood and used.

2

Background

2.1 Process Industries

The process industry is an umbrella term that includes a wide variety of manufacturing businesses that rely on highly controlled processes to transform raw materials into valuable goods. Examples of business areas that fall into the category are chemicals, pharmaceuticals, food & beverage, oil & gas, and the pulp & paper industry[5].

2.1.1 Physical Systems in the Process Industry

Plants in the process industry typically consist of a wide variety of components. Among the most common are tanks, pipes, pumps, and valves. Instrumentation measuring parameters such as pressure, temperature, flow, and level keeps the control system informed about the current state of every part of the plant. The complexity and size of these plants largely depend on the scale and complexity of the process that the plant is designed for. The sea of books, handbooks and guidelines for designing processing plants is nearly infinite. However, every plant depends on the right substances being in the right place, in the right quantity, at the right time. The control system that makes this happen relies on accurate information from sensors and other measuring equipment. An inaccurate level measurement may result in pumps running dry, too much of a substance in the new container or even overflow.

2.1.2 Tank Gauging and Level Measurement

Radar-based tank gauging and level measurement was first developed in the 1980s, and has since then been the technology of choice due to its high accuracy, lack of moving parts, and low maintenance requirements. Today, there are two key types of radar level measurement:

- Non-Contacting Radar - Emits microwaves that reflect against the surface of the product with no direct contact with the material.
- Guided Wave Radar - Uses a probe to guide the microwaves to the surface, making it suitable for low-dielectric materials or challenging tank geometries.

A typical tank gauging setup is shown in Figure 2.1.

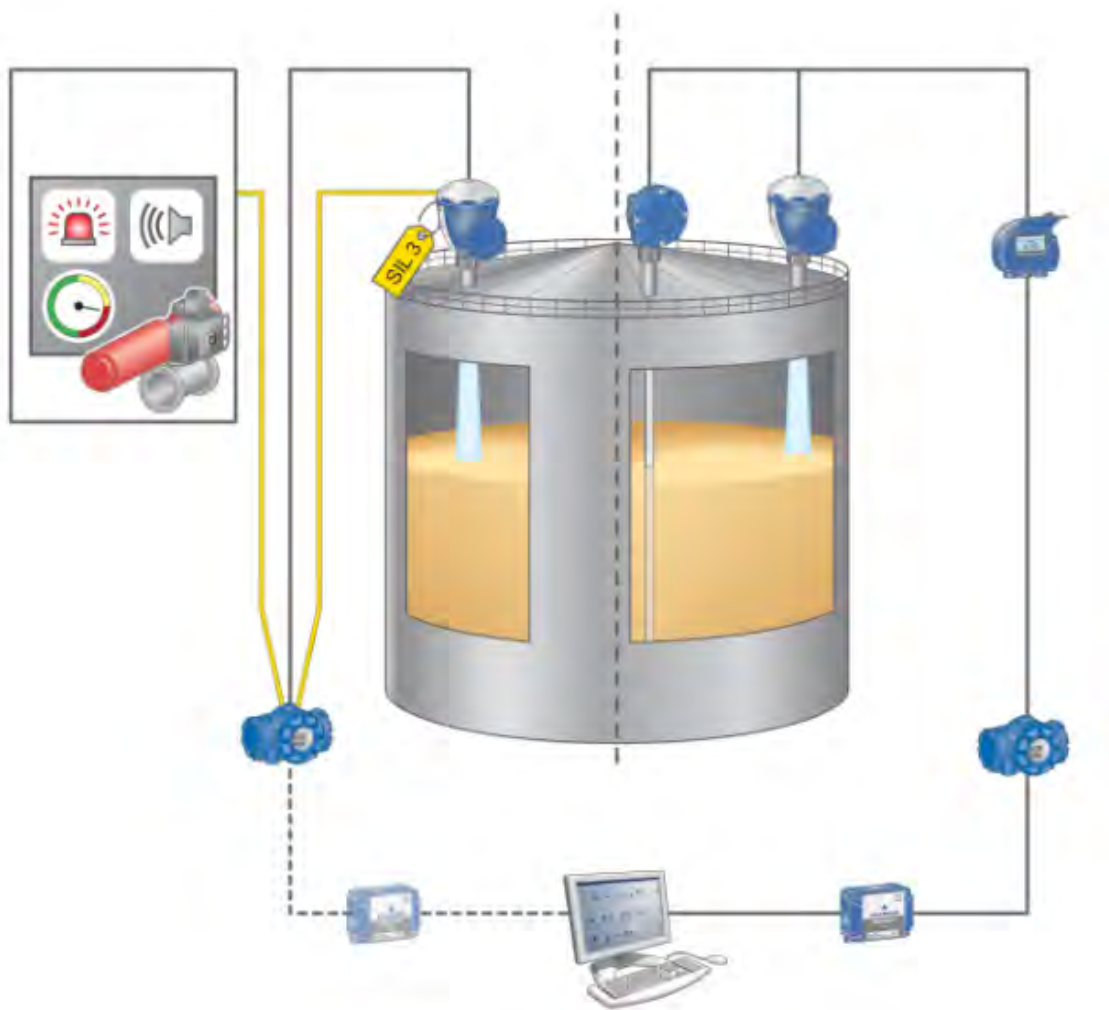


Figure 2.1: Illustration of a tank gauging setup.

2.1.3 Distributed Control Systems

The process industry has seen a digital transformation as the Distributed Control System (DCS) emerged as an orchestrator of modern process plants. A DCS is an industrial automation architecture that decentralizes the control and supervision of processes across autonomous units, connected through communication networks[6]. Local controllers handle real-time regulation, monitoring and optimization tasks, ensuring smooth operation. Every DCS relies on up to date, accurate information about the current state of the system that comes from measurement instrumentation[6].

2.2 Emerson (Rosemount Tank Radar AB)

This project is conducted at Rosemount Tank Radar (RTR), a division of Emerson Electric Co[7]. RTR develops and manufactures high-precision, non-contact radar level measurement solutions used across four primary market segments, as illustrated

in Figure. 2.2. These systems provide accurate and reliable level gauging together with integrated overflow protection and are deployed in both onshore and marine environments. While the markets share many technological and operational similarities, this thesis primarily focuses on applications within the process industries segment.

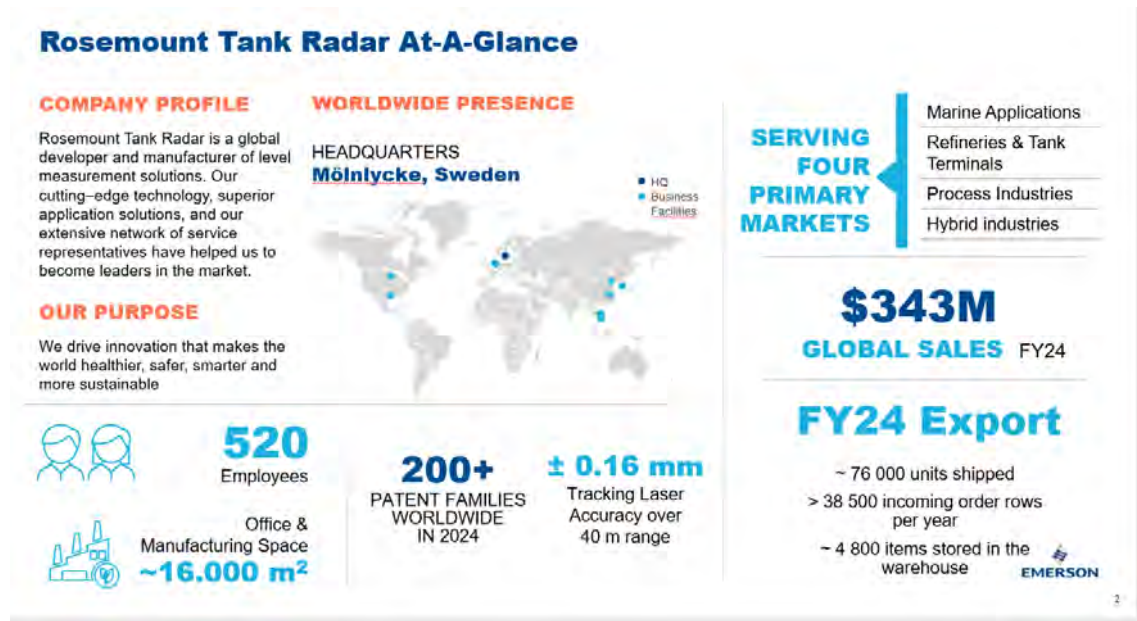


Figure 2.2: Summary of Rosemount Tank Radar

2.2.1 Interactive Plant Environment at Emerson

Currently, Emerson utilizes an Interactive Plant Environment (IPE), which is a mock process industry plant that is currently being used for sales and marketing purposes. However, the system is now being extended to support formal engineering work. Specifically, the Quality & Product (QP) department, which is responsible for system verification and testing, requires a functional software solution to execute test procedures and analyze results within the IPE. The mock setup consists of a series of tanks, for different liquids, interconnected via pipes. Valves and pumps control the flow in the pipes, and each tank is fitted with level measurement devices that provide a reference measurement for the liquid level. There is hardware in place for the DCS to use, and it is a small-scale representation of what a typical system "in the field" could look like. To successfully transition from a purely promotional tool to a functional engineering platform, the IPE requires a software solution to execute test procedures and analyze results.

2.3 Related and Existing Work

This section describes existing work and related work. Existing work in this context refers to solutions and software in the ecosystem at the company. Related work refers to similar systems.

2.3.1 Related Work

Several visual analytics systems have previously been developed for analyzing measurement and test data. Y. Zhao et al. present a visual analysis interface for exploring test outcomes, demonstrating the value of dedicated visualization for understanding complex test results[8]. Furthermore, IRVINE is a visual analytics system for analyzing acoustic measurement data from electric motors to detect manufacturing errors[9]. Additionally, R. Splechtna et al. present an interactive visualization system for high-dimensional spectral measurement data from automotive noise simulation. It uses multiple linked views to let engineers explore measurement data at different levels of detail[10]. Lastly, T. Funken and T. Meisen present a visual analytics application for analyzing industrial sensor time-series data[11]. However, it should be noted that these works are only concerned with data visualization, and not with designing an interface for setting up or running the actual tests.

Further research highlights the critical role of Information Visualization in complex engineering environments. Sindiy et al. demonstrate that applying InfoVis techniques to Model-Based Systems Engineering (MBSE) is essential for communicating complex architecture to stakeholders. They emphasize that simply producing technically correct models is insufficient if the visual presentation is poor, instead, interfaces must utilize techniques such as grid alignments, selective color palettes, and dynamic view generation to reduce cognitive load. By allowing users to interactively hide or show system components and automatically redrawing the views, engineers can rapidly answer specific questions about the system without risking corruption to the underlying model data. This aligns closely with the goals of VIPER, which similarly seeks to translate complex, underlying system data into an interactive and accessible visual format[12].

Research by Fekete[13] demonstrates that standard software toolkits lack native support for crucial information visualization mechanisms, such as dynamic queries, spatial deformations, and the dynamic mapping of abstract data to visual attributes. To achieve the fast action and feedback loops required for interactive data exploration, visualization tools must implement specific, optimized data structures[13]. This reinforces the necessity of custom-built interfaces, such as the Viper, which requires tailored architecture to properly manage, filter, and visualize high-dimensional industrial telemetry.

2.3.2 Existing Work

While Emerson has previously addressed test execution for different environments, those existing solutions are tightly coupled to their specific setups and cannot be reused for the IPE without significant modification.

In the existing in-house software suite, developed by Emerson, the following tools exist:

Instrument Manager (IM) is a tool that acts as an orchestrator for lab equipment, it handles communication, monitors, and is able to send commands to the equipment

it supports. Among other things, IM implements a GUI for a similar but outdated setup that Emerson is looking to transition away from.

Lemur is used for data collection and logging, and can be configured via scripts. As a result, this project will take advantage of existing Lemur software for data collection and logging in the IPE.

3

Theory

3.1 User Experience

The International Organization for Standardization (ISO) defines user experience (UX) as "A person's perceptions and responses that result from the use or anticipated use of a product, system, or service"[14]. UX design typically includes user research, creating personas, designing wireframes, creating interactive prototypes, and testing designs. While usability fits under the UX umbrella, UX goes beyond normal usability aspects by incorporating the users' feelings and emotions towards the product before or during interaction [15].

3.1.1 Interaction Design

Interaction design (IxD) is the design of interactive products and services, where the designer considers how users will interact with a product based on their needs, limitations and contexts[16]. An interaction designer considers every way a user will interact with a product, system, or service. IxD is said to consist of five dimensions [16]:

- Words (1D) encompass text, such as labels that gives users the right amount of information.
- Visual representations (2D) are graphical elements, typically images and icons that aid in user interaction.
- Physical objects/space (3D) typically refers to the hardware the user uses to interact with the product, system, or service, such as a mouse or keyboard for a computer.
- Time (4D) is media that changes with time, such as animations, videos, and sounds.
- Behavior (5D) is the combination of the previous four dimensions and how they define and shape the interactions.

3.1.2 Usability

Usability is a core concept in HCI and UX, concerned with how effectively, efficiently, and satisfactory users can achieve their goals when interacting with a system. Ac-

cording to the ISO 9241-11 standard, usability is defined as "the extent to which a system, product, or service can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use"[17].

According to the same standard, usability is always tied to the context, meaning it depends on the specific users, their goals, the tasks they perform, and the environment in which the interaction happens. Usability is therefore not fixed or universal but needs to be assessed in the context of actual use. HCI literature also shows that usability is influenced by factors such as learnability, memorability, and error handling, which describe how easily users can begin using a system, return after a break, and recover from mistakes[18].

Usability is also connected to user experience. While UX includes more than task performance, usability still plays a key role in shaping it. Interfaces that are difficult or frustrating to use lead to negative user experiences, whereas smooth and efficient interaction tends to support more positive ones[19].

3.2 Accessibility Standards and Guidelines

The World Wide Web Consortium (W3C) is an international, nonprofit organization that develops standards for the web. It was founded in 1994 by Tim Berners-Lee, the inventor of the World Wide Web, with the aim of supporting the longterm development of the web. Through collaboration between industry, academia, and public organizations, W3C works to create open standards that promote interoperability, accessibility, and consistent use of web technologies such as HTML and CSS[20].

One of the key standards developed by the W3C is the **Web Content Accessibility Guidelines (WCAG)**, which provide widely accepted recommendations for making digital content accessible to people with disabilities. WCAG is developed within the W3Cs **Web Accessibility Initiative (WAI)** through a collaborative, consensusbased process involving experts from different fields and regions[21].

Although WCAG is not a legal requirement in itself, it is widely used by governments and regulatory bodies as the technical basis for accessibility legislation. Due to this widespread adoption, WCAG is commonly regarded as the international reference standard for digital accessibility[21].

In addition to the WCAG standard, the W3C provides the Understanding WCAG documents. These documents offer explanatory, nonnormative guidance that clarifies the intent of each guideline and success criterion, and they are intended to support designers and developers in applying WCAG in practical design situations[22].

3.3 Information Visualization

Information Visualization (InfoVis) is the study and practice of transforming abstract data into visual representations that utilize the human visual system to facilitate understanding, reasoning and decision-making. Card, Mackinlay, and Shnei-

derman describe the goal of InfoVis as using interactive graphics to "amplify cognition", enabling users to discover relationships through visual thinking rather than purely symbolic processing[23]. A central theoretical pillar of InfoVis is the study of graphical perception, the work of Cleveland and McGill demonstrates that different encodings lead to a significant difference in perceptual accuracy[24]. Their ranked ordering where position on a common scale is decoded most accurately, followed by length, angle, area, volume, and color provides an empirical basis for selecting visual encodings in quantitative displays.

3.3.1 Methods of InfoVis

InfoVis offers a broad set of methods and techniques to represent abstract, non-spatial data visually. The methods revolve around visual encoding, interaction and layout principles that help users explore, understand and communicate complex information. Khan & Khan present several visualization methods including:[25]:

- Different types of pie charts
- Bar charts of varying types and styles
- Line and area charts
- Scatter plots

Wang et al.[26] present methods for filtering and settings, as well as how to provide details on demand for European power plant data. The user is given the option to choose which nodes to include and ignore as well as filter by various attributes. This lets the user create a customized view of the power system. The details are available on demand, and are shown separately from the "main" view.

In addition to these foundational methods, dealing with complex engineering data requires more specialized and interactive techniques. Some of these include:

- **Small Multiples:** A technique where similar graphs or images are placed side by side to allow users to easily compare data across different states or timeframes [12].
- **Fish-eye Lens:** An interactive method that provides a highly detailed area of focus while simultaneously maintaining the larger visual context of the surrounding information [12].
- **Dynamic View Generation:** An interactive approach that allows users to seamlessly add or remove elements from a diagram using high-level commands. The system then automatically redraws the visualization to match the user's immediate needs, which prevents cognitive overload by hiding irrelevant details[12].
- **Dynamic Labeling:** Providing labels dynamically based on pointer interaction, such as Excentric Labeling, to reveal details on demand without permanently cluttering the primary display[13].

- **Spatial Deformations:** Utilizing techniques like fisheye lenses or the Edge-Lens method to dynamically bend and push lines or overlapping edges away from the user's cursor, which drastically improves readability in dense data visualizations[13].
- **Dynamic Queries with High-Resolution Sliders:** Implementing fast filtering mechanisms using interactive components like range sliders. To handle large datasets, these sliders can utilize sub-pixel resolution, allowing users to make precise adjustments without the interface components consuming too much screen real estate[13].
- **Glyph-based representations:** Representing high-dimensional data as a unified geometric shape or glyph, allowing users to rapidly compare complex profiles at a glance[27].
- **Spring-based positioning:** Utilizing simulated physical models to dynamically place visual elements on the screen. Elements with similar attributes attract one another, automatically leading to the visual formation of data clusters[27].
- **Animation and Traces:** To process dynamic, time-series data, visualizations can utilize animation to represent state changes over time, while leaving visual "traces" to map the exact paths that specific data points take throughout the timeline[27].
- **Visual Analysis:** Visual analysis is an approach that combines automated analysis with interactive visualizations to understand complex datasets. For example, Zhao et al. demonstrate a visual analysis approach for understanding durability test data by separating the interface into a test process view and a channel view. As a specific method for data visualization within this approach, they present the centipede plot. This plot tackles the challenge of tracking an indicator over time by using a thick polyline to show the overall trend, combined with a band and directional lines to encode the degree and direction of local fluctuations[8].

3.4 Scientific Visualization

Scientific Visualization (SciVis) is concerned with the visual representation of scientific data, typically numerical, spatial, or simulation-based, to support understanding, hypothesis formation, and analytical reasoning. While InfoVis focuses on abstract data, SciVis typically deals with physically grounded, spatially embedded phenomena, such as fluid flows, molecular structures, and geospatial fields. According to Britannica, SciVis is the process of "graphically displaying real or simulated scientific data", often relying on computational techniques such as surface rendering, volume rendering, and animation[28].

3.4.1 Methods of SciVis

SciVis emphasizes methods that reflect the inherent structure of the data that typically come in the form of scalar, vector, tensor or volumetric fields, often defined over continuous spatial domains. Typical techniques include[29]:

- 2D Scalar Fields - color maps and contour lines to show gradients or level sets.
- 2D Vector Fields - glyphs, streamlines and line integral convolution to depict direction and magnitude.
- 3D Scalar Fields - volume rendering and isosurfaces allow scientists to explore internal structures of volumetric data.
- 3D Vector/Tensor Fields - particle tracing, streaklines and hyperstreamlines reflecting the flow or structural deformation of physical systems.

3.5 Wicked Problems

The concept of wicked problems originates from Rittel and Webber's work *Dilemmas in a General Theory of Planning* (1973). They argue that many issues encountered in social planning differ fundamentally from the structured and well-defined problems typically addressed in science and engineering. Wicked problems are "ill-defined" and cannot be understood through technical reasoning alone because defining them already shapes the path toward their solution[30].

Rather than having a clear "stopping rule" where a final, objective answer is found, work on a wicked problem typically ends only when time or resources are exhausted. Furthermore, solutions to these problems cannot be strictly judged as "true" or "false." Instead, they are evaluated as "better" or "worse" depending on the values, perspectives, and practical needs of the stakeholders[30].

Rittel and Webber emphasize that wicked problems are highly context-dependent. Because interventions carry real-world consequences and cannot easily be undone through trial and error, solutions cannot simply be copied from one unique context to another. There is no predefined set of permissible approaches, meaning that classical, linear systems-engineering procedures are fundamentally insufficient for resolving them[30].

In the context of this project, designing an interactive visualization system for the IPE exhibits several characteristics of a wicked problem. There is no single, objectively correct way to visualize complex telemetry data, and every design intervention shapes how test engineers interact with the system. Because the problem cannot be solved through linear optimization, it requires an iterative approach where the understanding of the engineers' workflow and the software solutions develop simultaneously.

3.6 Personas

Personas allow for a precise way of thinking and communicating about how groups of users think, behave, and what they want to accomplish. A Persona is a model that ideally emphasizes the important/significant features of the user group it represents, and de-emphasizes the less significant features[31]. According to Cooper[31], personas help **determine** what a product should do and how it should behave. They help **communication** between stakeholders, developers, and other designers. They are a common language for discussing design decisions. Additionally, they **build consensus and commitment** to the design as a common language brings common understanding. They also **measure** the design's effectiveness since design choices can be tested on a persona in the same way that they can be shown to a real user.

There are several traps that personas help avoid. A common pitfall is self-referential design, where designers "design for themselves" and not for the user. Personas are effective because they engage the empathy of the design and development team around the users' goals. This is critical for designers as they will make decisions about design frameworks and details based on the persona. *The elastic user* is another pitfall that personas help avoid, "the user" can stretch to fit whoever is talking, in other words the user is elastic. Personas always stay the same, and can not stretch to fit whoever is talking, providing precision about the user and clarity for the design team[31].

3.7 The 5 user guideline

Nielsen and Landauer propose that the discovery of usability problems follows a probabilistic pattern that can be modeled as a Poisson process. Their model predicts how the proportion of usability issues identified increases as more participants are tested. In this model, each participant has the probability of encountering any given usability problem, with empirical findings indicating a typical value of approximately 31%[32].

The model is expressed as follows:

$$\text{Found}(i) = N \left(1 - (1 - \lambda)^i \right)$$

where N represents the total number of usability problems in the system, λ is the average probability that a single participant will discover a given problem, and i denotes the number of participants included in the usability test[32].

Building on the usability problem discovery model, Nielsen argues that usability testing reaches diminishing returns as more participants are added. According to his analysis, the first few users uncover the majority of usability issues, while additional users tend to repeat the same findings rather than reveal new problems. After the fifth participant, additional sessions generally yield little new insight[33].

Nielsen therefore recommends conducting small, iterative rounds of usability testing. Instead of running single large test with 15 participants, he suggests dividing the same resources into multiple rounds with five participants, enabling the design to be improved and re-evaluated after each test cycle. This iterative approach typically identifies approximately 85% of all usability issues in the first round of testing, while subsequent rounds uncover most of the remaining problems, ensuring broader coverage[33].

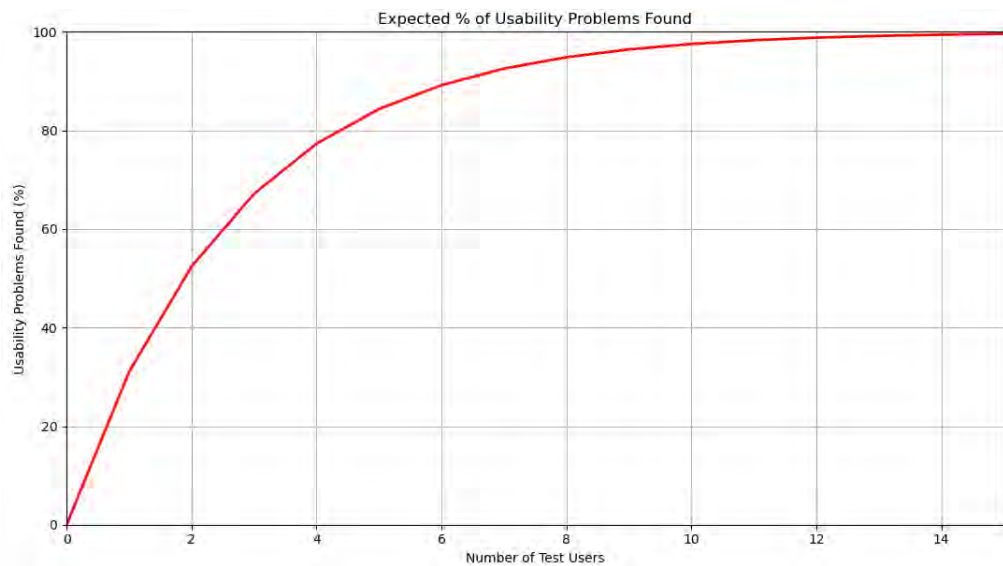


Figure 3.1: Expected proportion of usability problems found as a function of the number of test users.

4

Methodology

4.1 Research Through Design (RtD)

Research Through Design (RtD) is an approach that uses design as a means of conducting research and generating knowledge. Instead of just applying existing theories to a project, RtD uses the design process itself to explore a problem, letting new insights naturally emerge from building and testing prototypes. This is especially useful for "wicked problems", messy issues that are hard to accurately model or solve with traditional science and engineering methods[34].

RtD usually involves building an artifact or prototype. This artifact does not have to be finished product or a complete solution, instead it acts as a tool to uncover new insights. By following the approach, designers combine different areas of research to try and make the "right thing", a design that moves a situation from its current state to a better state. The main goal here is to create knowledge that helps both researchers and practitioners, rather than just building a commercial product to sell[35].

A big part of the RtD process is documenting decisions, iterations and reflections so the resulting knowledge is concrete and trackable. To keep a high quality research, researchers need to provide enough detail so the process can be reproduced and clearly explain why they chose their specific methods[34].

4.2 Double Diamond

The Double Diamond is a design process model popularized by Design Council in 2004. The model describes how designers move from exploring the problem space to delivering a final solution through two structured cycles of divergence and convergence. Divergent thinking involves expanding perspective, gathering insights, and exploring multiple possibilities, while convergent thinking focuses on narrowing the scope, integrating information, and making decisions. This alternation is crucial to the model and helps designers avoid jumping to conclusions too early, ensuring that the problem is properly understood before solutions are developed. The model consists of four different phases, grouped into two diamonds[36].

An extended version of the model is the **Triple Diamond** design process. This version adds an additional diamond, providing extra phases that allow for more

detailed exploration, evaluation, and refinement of ideas. Designers suggest that the Triple Diamond offers greater stability, giving teams more time and structure to address complex problems effectively[37].

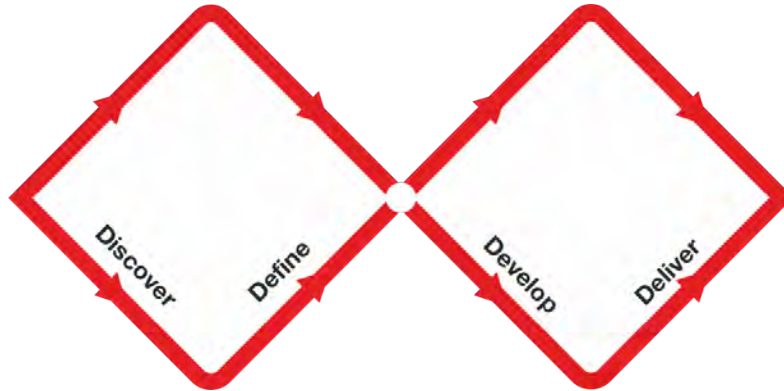


Figure 4.1: The Double Diamond by the Design Council [38].

4.2.1 Four phases of the Double Diamond

The Double Diamond is not a fixed or uniform process. The scale and emphasis of each phase can vary depending on the design challenge and project context. As a result, some phases may be more extensive than others, allowing the model to adapt to different needs while retaining its core structure of divergence and convergence [36].

4.2.1.1 Discover

The Discover phase is dedicated to developing insights into the problem space. Typical methods used here include interviews, field studies, and desk research[39]. The focus of this phase is on understanding the people affected by the issue, rather than relying on assumptions about what the problem might be[36].

Questionnaires. Questionnaires are a method of data collection in which a researcher presents a set of standardized questions to participants in order to obtain information that is relevant to the research topic. The researcher uses this instrument to collect clear data that can be systematically analyzed. **Closed questions** restrict participants to a fixed set of predefined options, which makes the data easy to quantify and compare among respondents. This format is mainly used to gather clear and consistent information, and it is especially useful when the aim is to analyze the responses statistically. **Open questions** allow participants to answer in their own words rather than selecting from fixed options. The goal is to gather rich and in depth qualitative data without restricting what participants can write about, which makes this approach useful for exploring complex issues where predefined categories are not enough[40].

Interviews. Interviews are a method of data collection in which a researcher asks a participant questions in order to obtain information that is relevant to the research topic. The researcher uses the conversation to obtain relevant insights based on the

participants experiences, knowledge, or perspectives. **Structured interviews** use a fixed set of questions that are asked in the same way to every participant, and the interviewer follows a strict script, which makes the answers easy to compare among participants. This format is mainly used to collect clear and consistent information such as demographics, user knowledge, or opinions, and it is especially useful when the aim is to ensure that all participants respond under the same conditions[41]. **Semi-structured interviews** use a set of predefined topics and questions, but the interviewer is free to adjust the order, skip questions, or add new ones when needed. This method is used when the main issues are already known, but more detail is needed and participants should have room to introduce new points[42]. **Unstructured interviews** are conversations guided by general topics rather than fixed questions, and there is no predetermined format since the interviewer adapts the discussion as it goes. The goal is to gather rich and in depth information without restricting what participants can talk about, which makes this approach useful for exploring new topics, capturing the views of the participants, and gaining insight into expert reasoning[43]. **Focus group interviews** involve a group discussion led by a moderator who guides participants through questions and activities about a specific topic. The moderator controls the discussion and participants share their views with each other in the group setting, and focus groups are used to gather a range of ideas, opinions and reactions of a certain topic[44].

	Structured interview	Semi- structured interview	Unstructured interview	Focus group
Fixed questions	✓	✓	✗	✓
Fixed order of questions	✓	✗	✗	✗
Fixed number of questions	✓	✗	✗	✗
Option to ask additional questions	✗	✓	✓	✓

Figure 4.2: Comparison of interview types based on question structure and flexibility [45].

Sampling Strategy. Sampling strategy refers to the approach used to select participants or data sources for a study. In research, sampling strategies are generally divided into **probability sampling** and **non-probability sampling**, each serving different purposes depending on the research design[46].

Probability sampling includes methods such as random sampling, systematic sampling, stratified sampling, and cluster sampling. These approaches ensure that every individual in a population has a known or equal chance of being selected, which supports statistical generalization[46].

Non-probability sampling relies on selecting participants according to researcher judgment, availability, or relevance. Common strategies include convenience sam-

pling, snowball sampling, quota sampling, and purposive sampling. These methods are particularly suited to qualitative research, where the aim is to generate rich and detailed insights rather than generalizable data. **Purposive sampling** in particular, involves selecting individuals who possess relevant characteristics or experience. It is useful when the goal is to obtain information that cannot be acquired through random selection alone[46].

4.2.1.2 Define

In the Define phase, insights and data from the Discover stage are analysed to identify the core problem and highlight key patterns. These findings are refined into a clear design plan that prioritises the main challenges and provides direction for the development of feasible and relevant solutions[36].

Thematic analysis. Thematic Analysis is an approach for working with qualitative data that provides a systematic yet flexible way to identify, analyze, and report patterns within a dataset. This approach is build around a set of analytic phases that goes from raw data into themes. Themes represent recurring and meaningful patterns identified across the data, and their purpose is to group these patterns into a form that supports the final interpretation[47].

The six phases include[47]:

- Familiarizing with the data
- Generating initial codes
- Searching for themes
- Reviewing themes
- Defining and naming themes
- Writing up

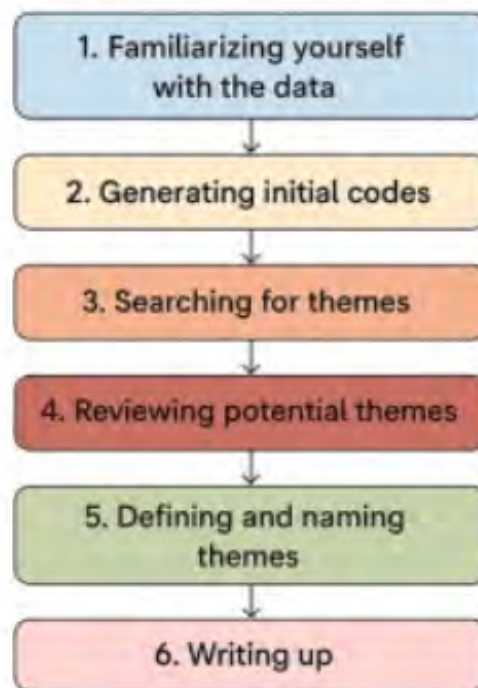


Figure 4.3: The six phases of Thematic Analysis[47].

Crazy 8. Crazy 8's is a rapid ideation method in which participants sketch eight distinct ideas, dedicating exactly one minute to each concept, in order to generate a wide variety of solutions relevant to a specific challenge. This exercise is designed to push participants beyond their initial, most obvious concepts. One-minute intervals restrict participants to a strict time limit per sketch, which forces a rapid generation of visual concepts and prevents overthinking. This format is mainly used to collect a large quantity of rough concepts quickly, and it is especially useful when the aim is to focus on communicating the core idea rather than creating polished drawings[48].

Storyboarding. Storyboards are a visual storytelling method used to predict and illustrate how a user will experience a product over time. A storyboard is a series of drawings that shows what the user does, where they are, and how they use the system. According to Nielsen Norman Group[49], storyboards help visualize the flow of an experience and expose usability issues early in the design process. They provide a simple timeline for discussing how tasks happen step-by-step, ensuring that design decisions look at the whole situation and not just the screen itself.

There are several traps that storyboards help avoid. A common pitfall in digital design is ignoring the user's physical surroundings and focusing entirely on the interface. Storyboards are effective because they force the design team to think about outside factors, such as where the user is or if they are distracted. This is critical for designers as they will make decisions about the user journey based on these everyday situations. Finding missing steps or mistakes in the interaction early on prevents the team from spending time on flawed prototypes, providing a cheap and easy way to test the system's logic[49].

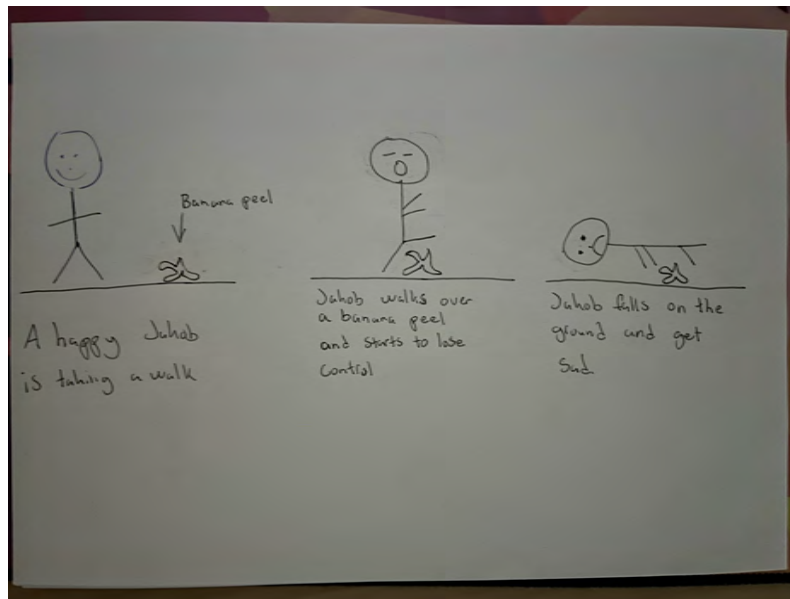


Figure 4.4: A basic three panel storyboard illustrating a user scenario.

Six Thinking Hats. The Six Thinking Hats method, created by Edward de Bono in 1985, is a structured brainstorming technique that changes how groups think together. Instead of the usual setup where people argue back and forth over an idea, this method uses something called "parallel thinking." This simply means that everyone in the group looks at the problem from the exact same angle at the exact same time. It is used primarily during the early stages of a project when a team needs to generate ideas, solve a complex problem, or make a decision. It is especially helpful when a group is stuck, arguing in circles, or when a few dominant voices are taking over the conversation. The biggest advantage of this method is that it takes the ego out of the equation. Because participants are just "roleplaying" a specific type of thinking, they do not get defensive when an idea is criticized. For example, a naturally pessimistic person is forced to look for the good in an idea, and a super optimistic person is forced to pause and look at the risks[50].

The Six Hats and What They Mean. To make this work, the method divides the thinking into six different styles, represented by six colored hats. During a session, the team "wears" one hat at a time and only focuses on that specific type of thinking, with the exception of the Blue Hat, which is usually worn by the facilitator throughout the entire process [50]:

- **White Hat (Facts):** This hat is all about objective data. The group looks at what information they currently have, what facts they are missing, and how to go out and get them.
- **Yellow Hat (Optimism):** This is the positive hat. The team actively looks for the benefits, the value, and the reasons why an idea could work.
- **Black Hat (Caution):** This is where the team actively looks for potential problems. It is all about spotting the dangers, pointing out flaws, and figuring out exactly what could go wrong with an idea in the real world.

- **Red Hat (Feelings):** This hat is about emotion and intuition. People can share their gut reactions, fears, or hunches about an idea without having to explain or justify them with logic.
- **Green Hat (Creativity):** This is the generative hat. It gives everyone the opportunity to express creative concepts and look at the problem through new perceptions.
- **Blue Hat (Control):** This hat is about managing the process. It is usually worn by the person leading the meeting to keep track of time, decide which color hat the group should wear next, and summarize the final results.

4.2.1.3 Develop

The Develop phase focuses on generating, testing, and refining potential solutions. Teams explore ideas through sketches and prototypes, collaborating with users and other specialists[36]. This phase emphasizes divergent thinking, experimentation and iterative trial and error to improve and refine concepts[51]. Activities include brainstorming, prototyping and testing, all of which contribute to evaluating alternative approaches and shaping feasible and innovative solutions for the Deliver phase.

Prototyping. Prototyping is a core activity in human-centered design and is used to explore, refine, and develop design ideas. It helps designers concretize early concepts, evaluate alternatives, and iteratively improve a solution before full implementation. Prototypes vary in fidelity depending on their purpose, ranging from rough sketches to detailed, interactive representations of the final product [19], [52].

Low-fidelity prototypes. Low-fidelity prototypes are quick, simple, and inexpensive representations of a design. They focus on structure, layout, and basic interaction flow rather than visual detail, supporting rapid exploration and early feedback [52]. Common low-fidelity techniques include sketches and wireframes. Sketches help generate and explore early design ideas without focusing on detail, enabling fast iteration in the early stages of design [52]. Wireframes extend sketches by providing a more structured representation of the interface layout and information hierarchy, clarifying how users will interact with the system and serving as a bridge toward higher-fidelity prototypes [19].

High-fidelity prototypes. High-fidelity prototypes are detailed representations that aim to match the final product in terms of appearance, interaction, and behaviour. They allow for more accurate user testing because they provide a more complete and interactive experience [19]. A common high-fidelity technique is the interactive mockup, which simulates the behavior and navigation of the final system by connecting screens and interface components. Unlike static wireframes, interactive mockups allow users to perform realistic actions such as clicking, navigating, or completing tasks, making them useful for evaluating whether the interaction flow works as intended prior to implementation [19].

4.2.1.4 Deliver

The final stage of the Double Diamond model, focuses on finalising and launching the product or service. Concepts undergo final testing, approval, and refinement before release to ensure that only viable solutions are delivered[36]. Systems are put in place to capture user feedback and monitor outcomes, ensuring the solution performs as intended. This phase also includes encapsulating key insights and lessons from the process for future projects[51].

Testing. Testing in the Deliver phase focuses on validating whether the final prototype or solution performs as intended and meets user needs before implementation. **Usability testing** is commonly used to evaluate a product or service by observing users as they perform representative tasks. The aim is to identify usability problems, collect qualitative and quantitative data, and determine the participants satisfaction with the product, focusing on the users experience and their ability to complete tasks efficiently and effectively[53]. One widely used approach within usability testing is the **Think Aloud** method, where participants verbalize their thoughts, feelings, and opinions while performing tasks. This allows researchers to capture the users decisionmaking process in real time and understand not only where a problem occurs, but why it occurs[18]. Think Aloud can be conducted either **concurrently** or **retrospectively**. In concurrent Think Aloud, participants speak while performing the task, providing immediate insight into user focus but sometimes increasing task completion time due to the added cognitive load[54]. In retrospective Think Aloud, participants complete the task silently and then describe their thoughts afterward while watching a recording of their session, avoiding interference with task performance but relying on memory, which may be less accurate[55].

4.2.2 Evaluation Framework: Heuristics and Principles

Embellished frameworks like Nielsen’s Usability Heuristics and Shneiderman’s Eight Golden rules provide a strong baseline for interface design. While these methodologies offer a structured approach to assessing general software, they are often not enough to capture the complex requirements of specialized, real time industrial verification systems. Therefore, in this project, these frameworks are not used as a strict evaluation checklist. Instead, they serve as a theoretical baseline. By combining these core usability principles with Information Visualization theory and insights from our user interviews, this project introduces a custom set of design guidelines specifically for industrial equipment verification.

Nielsens 10 Usability Heuristics [56]

Nielsens heuristics are a set of broad principles used to identify common usability issues in software interfaces. By focusing on how users naturally interact with computers, these guidelines help ensure that a system is predictable, transparent, and easy to navigate. They are particularly effective for complex systems that require high levels of user awareness. Instead of providing strict visual rules, these heuristics evaluate how well an interface provides continuous feedback and matches the user’s real-world expectations.

Heuristic	Description
1. Visibility of system status	The design should always keep users informed about what is going on, through appropriate feedback within a reasonable amount of time.
2. Match system and real world	The design should speak the users' language. Use words, phrases, and concepts familiar to the user, rather than internal jargon. Follow real-world conventions, making information appear in a natural and logical order.
3. User control and freedom	Users often perform actions by mistake. They need a clearly marked "emergency exit" to leave the unwanted action without having to go through an extended process.
4. Consistency and standards	Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform and industry conventions.
5. Error prevention	Good error messages are important, but the best designs carefully prevent problems from occurring in the first place. Either eliminate error-prone conditions, or check for them and present users with a confirmation option before they commit to the action.
6. Recognition rather than recall	Minimize the user's memory load by making elements, actions, and options visible. The user should not have to remember information from one part of the interface to another. Information required to use the design (e.g. field labels or menu items) should be visible or easily retrievable when needed.
7. Flexibility and efficiency of use	Shortcuts hidden from novice users may speed up the interaction for the expert user so that the design can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.
8. Aesthetic and minimalist design	Interfaces should not contain information that is irrelevant or rarely needed. Every extra unit of information in an interface competes with the relevant units of information and diminishes their relative visibility.
9. Help Users Recognize, Diagnose, and Recover from Errors	Error messages should be expressed in plain language (no error codes), precisely indicate the problem, and constructively suggest a solution.
10. Help and documentation	It's best if the system doesn't need any additional explanation. However, it may be necessary to provide documentation to help users understand how to complete their tasks.

Shneidermans 8 Golden Rules [57]

Shneidermans rules aim to make interfaces more efficient and reduce the mental effort required from the user. This framework is especially valuable for systems that handle large amounts of data. Instead of focusing on data visualization, these rules [57] emphasize building predictable interactions, providing clear feedback, and

4. Methodology

allowing easy error recovery. For verification tasks, this structured interaction prevents cognitive overload, ensuring users remain in control and can safely navigate the system to identify errors.

Golden Rule	Description
1. Strive for consistency	Use consistent sequences of actions in similar situations and identical terminology in prompts, menus, and help screens.
2. Seek universal usability	Recognize the needs of diverse users (novice vs. expert, age ranges, disability) and design for plasticity.
3. Offer informative feedback	For every user action, there should be appropriate system feedback. Minor actions get modest responses; major actions get substantial ones.
4. Design dialogs to yield closure	Organize sequences of actions into groups with a beginning, middle, and end to give users a sense of accomplishment and relief.
5. Prevent errors	Design the system so users cannot make serious errors. If they do, the system should offer simple, constructive instructions for recovery.
6. Permit easy reversal of actions	Actions should be reversible to relieve anxiety and encourage exploration of unfamiliar options.
7. Keep users in control	Experienced users want to feel that they are in charge of the interface and that it responds to their specific actions.
8. Reduce short-term memory load	Keep displays simple, consolidate multiple page displays, and reduce window-motion frequency to respect human memory limits.

5

Process

5.1 Project Planning and Execution

Towards the end of the pre-study, a preliminary timeline for the project was created, see fig. 5.1. The purpose of this timeline, other than serving as a timeline visualization, is to align internal and stakeholder deadlines. Some phases overlap as they can be progressed simultaneously.



Figure 5.1: Gantt chart - planned process for project

However, as we moved out of the research phase and started actively writing code, our workflow naturally evolved. This second chart shows our actual timeline. and how we adapted to real technical challenges and shifted our priorities.



Figure 5.2: Gantt chart - actual process for project

The main difference between the two charts is the length of the implementation phase, which took much longer than initially expected due to the implementation being much more complex than we expected. However, as the project progressed, we realized that the tasks following implementation would either take less time or could overlap with each other. This understanding allowed us to intentionally dedicate more time to the implementation phase.

5.1.1 Project Phases

- **Requirements Gathering** - Conduct semi-structured interviews with QP department members, sales & marketing personnel and Q/QP leadership. The output of this phase should be a requirements specification that contains must-have features for testing/validation use, must-haves for demonstration/showcase, overlapping features and nice-to-have features for both purposes.
- **Design Phase** - Employ a user-centered design methodology with personas from both user groups, develop scenarios for the personas, and sketch a design that accommodates must-haves and enables the personas to carry out their tasks. The output should be a system architecture specification, interface wireframes and interaction flows.
- **Implementation** - Develop must-haves with prioritization based on stake-

holder input, either simulation capabilities or physical system integration as main focus. Expected technical components include communication layer with physical/simulated equipment, scenario engine for creating and running test cases, data logging and analysis capabilities, GUI with appropriate features. The output should be a functional prototype.

- **Evaluation** - Evaluate the prototypes usability with representative users from both groups, task-based evaluation and semi-structured interviews after using the prototype. Evaluate whether the dual-purpose design compromises any of the use cases. The output should be evaluation results documenting system effectiveness for both purposes.
- **Analysis** - The analysis should synthesize the findings, reflect on design decisions and their impact, document the design patterns and lessons learned.
- **Expected Output** - The project expects to result in a functional simulation system serving the needs of the Q/QP department in their testing and validation work but also the showcase and demonstration purpose. Expected research contributions include empirical insights into designing systems that serve both technical and non-technical users, understanding the trade-offs and synergies when designing for dual-purpose.
- **Limitations** - This project does not aim to develop novel simulation techniques or algorithms, create a production-ready system or provide exhaustive coverage of all possible field scenarios.

5.1.2 Scrum Framework

The project was planned to follow the Scrum framework, an iterative method where work is organized into short sprints that include planning, review, and retrospective activities. Scrum supports continuous feedback and enables the team to make adjustments as the project progresses[58]. In the context of this thesis, the Scrum framework was specifically adapted to suit a two person development team working in close collaboration. This methodology was chosen because one team member had prior professional experience working with agile frameworks and recognized its value for maintaining momentum, structure, and open communication.

Sprint Plan							Recurring events						
Mon	Tue	Wed	Thu	Fri	Sat	Sun	Standups: 9:00 every morning						
Sprint planning							Staffan meetings: 9:00 fridays at Chalmers						
Mon	Tue	Wed	Thu	Fri	Sat	Sun	Yuxuan meetings: thursdays after lunch						
		Update backlog	Sprint review	Sprint retrospective									
				update research question answer									

Figure 5.3: Sprint plan

The project was divided into two week sprints. To manage the workflow, track tasks, and maintain the backlog, the team utilized Trello as the primary project management tool. The two week sprints were structured around four key agile events:

Sprint Planning: At the start of each sprint, a planning meeting was held to define the upcoming workload. The meeting began by reviewing the calendar to account for any planned absences or special events, ensuring the team had a realistic understanding of their capacity for the next two weeks. New tasks were then introduced, broken down into manageable sub tasks, and given clear, detailed descriptions so both team members had a shared understanding of the requirements. Finally, tasks were moved from the main backlog into the sprint backlog, carefully balancing the workload to ensure it was challenging but realistic.

Daily Standups: To maintain daily alignment, brief synchronization meetings were held every morning at 9:00 AM, shortly after arriving at the office. These meetings focused on checking each member's current status, ensuring everyone had clear tasks for the day, and identifying any technical blockers where support was needed. It was also an opportunity to highlight any special events or meetings scheduled for that specific day.

Sprint Review: At the end of the two week cycle, a review meeting was conducted to evaluate the technical progress. During this meeting, each team member individually presented the specific tasks they had completed over the sprint to establish a shared understanding of the newly built components. Following this, any unfinished tasks were openly discussed to understand why they were delayed, whether the initial scope was too large, or if further collaboration was required. The meeting concluded by setting general expectations and technical goals for the following sprint.

Sprint Retrospective: Following the sprint review, normally the same day, a retrospective was held to reflect on the team dynamic, working process, and overall well being. The meeting started by mapping out a physical timeline of the sprint, marking down notable events on specific days (for example, a meeting with the company manager or a visit from the academic supervisor).

Team members then wrote down brief observations about the sprint and labeled them as "good," "bad," or "other." These notes included both practical project issues such as feeling stuck on a specific coding task or managing our time poorly, as well as general personal updates to keep communication open. To prioritize the discussion, we used a straightforward dot voting system where each person received three votes to place on any note. The notes with the most votes were then discussed so we could adjust our daily working habits for the following weeks.

academic foundation securely in place, the user research and interview preparations were officially pushed to the next sprint.

Sprint 3: User Research and Thematic Analysis (Weeks 5-6)

- **The Goal:** Plan and conduct the user interviews, and analyze the feedback to figure out exactly what the test engineers needed.
- **Execution:** We started this sprint by researching best practices for qualitative interviews and writing a semi-structured interview guide. After booking time with the test engineering team, we conducted the interviews to learn about their daily workflows and challenges. Once the interviews were finished, we immediately ran a thematic analysis on our notes. This allowed us to group all their feedback into clear, recurring patterns.
- **Outcome:** By the end of the sprint, we had completed all the interviews and finished the thematic analysis. This gave us a solid list of user needs, meaning we were fully prepared to start brainstorming and sketching interface concepts in the next sprint.

Sprint 4: Sketching and Technical Planning (Weeks 7-8)

- **The Goal:** Generate visual design ideas, pick a main interface layout, and decide on the programming tools and boundaries for the software.
- **Execution:** We started this sprint with fast brainstorming and sketching sessions, including the Crazy 8s method, to explore different interface ideas. We then evaluated these sketches using a sticker voting method, which helped us drop the weaker ideas and combine the best parts into one clear design. With the visual direction chosen, we set the project's technical boundaries and officially chose our programming languages and tools. Finally, we took our first steps into coding by making digital versions of our sketches to see how the ideas looked on a real screen.
- **Outcome:** A final design concept, a confirmed set of programming tools, and early digital drafts. This gave us a clear plan, finishing the exploratory phase and getting us ready to start building the actual software.

Sprint 5: Initial Coding and Prototyping (Weeks 9-10)

- **The Goal:** Start active development, build the core foundation of the software, and check our direction with our stakeholders.
- **Execution:** We moved from planning to actual coding, focusing on building the basic shell of the application. To test our design ideas on a real screen, we built static interface parts, like buttons and layouts that were not yet connected to the backend. Just to see how they looked and if the user flow made sense. Throughout this sprint, we worked closely with our company mentor and the test engineers. These frequent, informal check-ins helped us quickly refine our understanding of exactly what the final product needed to do.

- **Outcome:** We successfully built the basic software structure and, more importantly, found a highly efficient coding rhythm. Because our goals were now completely clear and our teamwork was seamless, we realized the formal sprint structure was no longer needed. This realization officially ended our exploratory phase, and we naturally transitioned into a continuous workflow.

5.1.2.2 Evolution of the Working Process

During the pre-study, data collection, and ideation phases, we followed the strict Scrum framework we described in Section 4.1. We worked in two week sprints and had daily standups and sprint planning meetings. This gave us a lot of structure when we needed to explore the problem and figure out exactly what to build.

However, a few weeks into the actual coding, we realized that we did not need all these formal meetings anymore. Since it was just the two of us working closely together all the time, booking meetings to plan our work started to feel like unnecessary overhead. We already had a strong shared understanding of the technical goals, so it was enough to just ask each other questions and talk about the work continuously. We decided to drop the formal sprints and shifted to a more fluid, flexible workflow for the rest of the project. This made our development pace much faster and more natural, though we still kept our regular weekly check-ins with our academic supervisor and company mentor.

5.1.3 Design Process

The design process for this project followed an iterative and user-centered approach inspired by the Double Diamond model. The work began with a pre-study and data collection phase, followed by a thematic analysis of interview data to identify challenges and user needs. Based on these insights, we formulated How Might We questions to guide the ideation phase. The ideation consisted of sketch based methods, including a Crazy 8 session and additional sketching, which resulted in a set of concept ideas. These ideas were then evaluated using a sticker voting method. The following sections describe each of these stages in detail.

5.2 Pre-study

The first four weeks of the project were dedicated to a pre-study aimed at building an initial understanding of the domain and shaping the overall direction of the work. During this period, a significant amount of literature was reviewed, primarily through Google Scholar and the Chalmers library databases. We also revisited literature and course material from earlier university studies to refresh key concepts and support the theoretical foundation.

Because the project was conducted at a company, the pre-study also involved understanding practical conditions and limitations. This included discussing what was realistic to achieve within the timeframe, which tools and data were available, and

what expectations the company had. These insights were important for adjusting the scope and ensuring the project remained feasible.

Insights from both the literature review and the company context helped define the problem space and guided the formulation of the projects goals, boundaries, and focus areas.

Theoretical Framing: During this pre-study, both Information Visualization (InfoVis) and Scientific Visualization (SciVis) were considered as foundational frameworks. Although they share many similarities, they emphasize different methods. For this project, the aforementioned methods of SciVis, scalar and vector fields, were deemed to not be the best fit for visualizing the IPE sensor data. Therefore, it was decided that InfoVis methods provided the more appropriate theoretical framework for the design process moving forward.

5.2.1 Interviews

The purpose of the interviews was to gain an understanding of the current testing workflow, identify recurring challenges, and capture expectations that could later be translated into design requirements.

Semi-structured interviews were conducted to explore the experiences, challenges, and expectations of the participants about the current testing workflow. This method was chosen because it allows followup questions while still providing structure, making it suitable for exploring workflow specific issues and clarifying ambiguous responses. Each interview lasted approximately 15 minutes and was conducted in a quiet meeting room at the company. Each interview was conducted by two researchers, one who led the conversation and another who took notes to ensure that key details were captured accurately.

5.2.1.1 Participant Selection

Five members of an internal software testing team at the company were invited to participate in the study. The group consisted of four test engineers and one team leader. Although the team leader no longer performs daily testing tasks, he previously worked as a test engineer and therefore has relevant knowledge of the workflow. One participant was newly employed at the time of data collection, which resulted in varying levels of experience within the participant group.

Participants were selected using purposive sampling, as the goal of the project is to design a solution intended specifically for this teams workflow. Participants represent the primary user group of the existing workflow, making their domain expertise essential to guide the design process.

Four participants provided their written consent to participate in the study. One participant declined to sign the consent form, but agreed to answer the questions informally. Following ethical procedures, data from the participant who declined written consent were omitted from both transcription and analysis. Only data from the four consenting participants were included in the studys results.

5.2.1.2 Interview Procedure

Each interview began with an introduction to the study and an explanation of the participants rights, including anonymity and voluntary participation. Participants were asked to sign a consent form. Audio recordings were not used due to practical constraints and to reduce potential discomfort among participants. The presence of two researchers helped mitigate the risk of missing important details. Instead, one researcher asked the questions while the second researcher took detailed notes on a computer to accurately document the participants responses. All notes were stored securely (as detailed in Section 1.4) and later used in the data analysis. A set of guiding questions was prepared in advance, but these were only used to guide the conversation. The interviewer adapted and added questions naturally based on the participants responses.

5.2.2 Thematic Analysis

The interview notes were analysed using a thematic analysis approach inspired by Braun and Clarke. The process began with familiarization, where the interview material was read through several times to gain an overview of the data. Key statements from the interviews were then written on individual Post-it notes, each representing a meaningful idea or observation made by the participants. These notes were visually arranged and grouped based on similarity, allowing patterns and recurring issues to emerge. Each group was then given a descriptive label that captured the essence of the statements it contained. Finally, short summaries were written for each group to clarify their meaning and to support the development of initial themes. These themes formed the foundation for understanding the main challenges and needs within the current workflow.

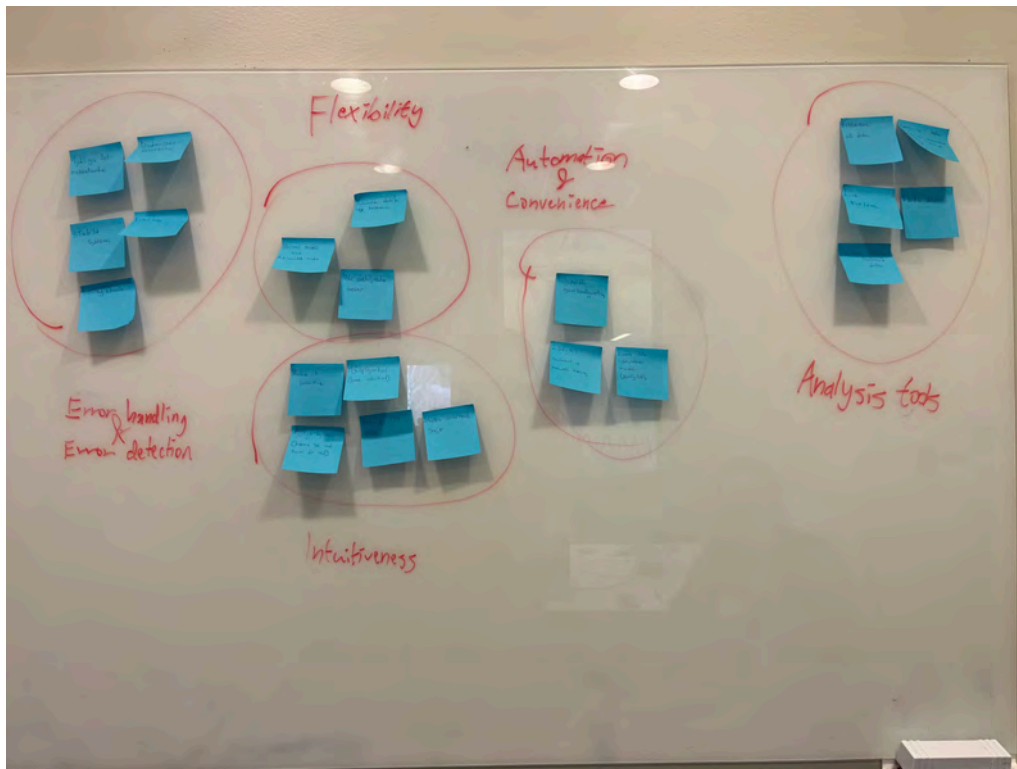


Figure 5.5: Grouping of Postit notes during the thematic analysis, showing how key statements from the interviews were clustered into early theme categories.

5.2.3 Themes from Interviews and Thematic analysis

This section presents the results of the interviews conducted. The interview data was analysed using thematic analysis, resulting in several recurring themes related to usability, flexibility, error handling, and analysis capabilities in current testing tools.

Theme 1: Limited Support for Error Handling The participants generally described the current system as stable and functional for executing both test scripts and analysis scripts. However, several limitations related to error handling were identified. Error messages are shown primarily on the console, making it difficult to understand why and when an error occurs. Participants expressed that missing or unclear error feedback slows down troubleshooting and increases frustration. The absence of error logs and sanity checks was highlighted as a recurring issue, as errors may only be discovered at a later stage.

Theme 2: Lack of Intuitiveness for New or Less Experienced Users Multiple participants described the system as difficult to understand for beginners. The interface assumes prior knowledge, and there is limited guidance when users encounter unclear input or configuration options. The participants noted that teaching and instructing new users takes time and that the system does not support learning through examples or templates. The lack of help symbols, explanations, or execution templates makes onboarding more difficult and increases the dependency on

prior experience.

Theme 3: Greater Flexibility in Test Execution Although scripted execution was considered efficient, the participants expressed a need for greater flexibility. Several users wanted the ability to execute specific parts of a test script, run tests in an advanced mode, or manually queue instructions without writing full scripts. The ability to switch between automated execution and manual control was seen as important, especially for exploratory testing or troubleshooting. Participants also expressed interest in combining different interfaces, allowing users to choose between script-based execution and a more interactive environment.

Theme 4: Insufficient Analysis and Visualization Capabilities The analysis tools were identified as a major limitation. Participants reported that they often need to manually plot or inspect data, which is time consuming and inefficient. The current tools provide limited information about what happens during test execution, offering mostly input and output without intermediate visibility. Desired improvements included:

- Zooming into specific time spans
- Adjusting axis ranges
- Live plotting
- Screenshots of data
- Clear visualization of specific test segments

Theme 5: Automation and Configuration Convenience The participants emphasized the importance of automation and convenience in testing setup and execution. Automatic device configuration was considered essential as repeatedly entering the same information was perceived as unnecessary and prone to errors. In addition, participants expressed a desire for smoother data handling, such as automatically saving test data in the correct location and reducing the need for manual file transfers or repeated setup steps.

5.2.4 Synthesizing the Design Guidelines

The thematic analysis provided a clear picture of the daily struggles test engineers face when verifying equipment. However, simply identifying these pain points is only half the process. To ensure these insights actually result in a better interface, they need to be translated into actionable design rules.

To do this, we looked back at established HCI frameworks, specifically Niensens heuristics and Shneidermans Golden Rules. While these rules are great for general software, our thematic analysis proved they are too broad for a live industrial environment. Because of this, we used the themes from our thematic analysis to tailor these foundational theories. We took the specific problems the engineers talked about and connected them to established HCI and InfoVis principles. This combination became our custom framework.

The result is a set of four domain specific design guidelines. These guidelines serve as the foundation for our Interactive Simulation System, ensuring that every design choice is backed by both established theory and real world engineering needs.

5.2.5 Initial Design Guidelines

By combining our thematic analysis, Information Visualization theory, and foundational HCI principles, we propose the following four design guidelines.

1. Match the Physical Setup

Guideline: The interface should visually represent the actual plant instead of just presenting numerical data on the screen.

Justification: Our interviews showed that the current system is difficult for new users to grasp (**Theme 2**) because making sense of the screen requires too much prior knowledge about how the data is structured. On top of that, **Theme 4** showed that engineers struggle to see what is actually happening inside the equipment during live test. Instead of getting a clear picture of the system's state, they are forced to manually decode raw numbers.

While the participants did not explicitly request graphical models, solving this requires a shift in how data is presented. Nielsen's heuristic to "Match the system and the real world" states that an interface should use concepts familiar to the user. Since the engineers are testing physical equipment, we propose that Information Visualisation principles must be applied to make the interface match the reality. If a system replaces raw data with visual representations, like tanks or pipes it would remove the need for users to decode numbers in their heads. We expect this visual approach to directly solve the visibility and learning curve issues that the engineers brought up.

2. Flexible Test Control

Guideline: The interface should allow users to easily switch between running automated test scripts and making quick manual adjustments.

Justification: Troubleshooting industrial equipment is rarely a simple step by step process. **Theme 5** showed that engineers need automated scripts for repetitive setups, but **Theme 3** proved that forcing them to use only automation actually slows them down. The engineers specifically asked for the ability to run manual commands and do exploratory testing without constantly writing new code.

To solve this, we took inspiration from Nielsens heuristic of "Flexibility and efficiency of use," which suggests that system should support different ways of working without getting in the user's way. We propose that a practical simulation interface needs to handle both automation and manual control without forcing a strict workflow. If the system lets an engineer run an automated test, but still allows them to pause it, jump in manually or force an override without restarting, then it eliminates that

restriction. This approach is designed to solve the workflow frustrations brought up in the interviews.

3. Stable Views for Live Data

Guideline: The interface must never auto scroll or force view changes when the user is trying to analyse a specific event in a live data stream. **Justification:** In normal software, if you miss something or make a mistake you can usually just hit the "Undo" button. But in a live industrial simulation, you can not simply pause or undo incoming sensor data without disrupting the physical test. **Theme 4** showed that the current analysis tools are a major restriction, because engineers have limited viability while a test is executing, and specifically requested the addition of live plotting and the ability to zoom into a specific time frame.

Because introducing live data creates a constant stream of moving information, we propose that view stability must be the priority. This aligns with Shneiderman's rule to "Keep users in control," ensuring the interface never surprises the user with sudden changes. If an engineer spots an anomaly and zooms in to look at the specific data spike, the system must not autoscroll or jump back to the newest incoming data point. The system should keep logging data in the background, but the user's screen must stay locked exactly where they want it until they chose to go back to the live plot.

4. Clear Error Handling

Guideline: The interface should provide a clear distinction between different types of system feedback, ensuring that errors are immediately recognizable and stored in a way that is easy for engineers to handle.

Justification: During the interview, **Theme 1** showed that the current way of working with errors is frustrating and not optimal. Since everything is outputted in the console, it is hard for engineers, especially if they are new to tell the difference between a serious error and an update message. On top of that, because the system does not log this for you, the engineers have to manually copy or save the data themselves, which is a slow and annoying way to work.

To fix this, we took inspiration from Nielsen's rule "Help users recognize, diagnose, and recover from errors." We propose that the interface should make errors stand out so they are harder to miss, while also providing automatic logging. If the system handles the part of catching and storing errors, the engineer does not have to worry about losing data or fighting with a scrolling console. This approach is designed to replace a manual and annoying process with a workflow that makes more sense for the user.

5.3 Ideation and Concept Generation

After completing the thematic analysis, we turned the main insights into a set of How Might We (HMW) questions. These questions helped us decide which directions to

explore during ideation. Based on the HMWs, we moved into a sketch-based ideation process to start generating potential solutions.

5.3.1 Sketching

We started the sketching phase by doing a Crazy 8 session to push ourselves to generate ideas. The exercise helped us get past the more obvious solutions and quickly put different concepts on paper. After the Crazy 8 session, we realized that we still had more ideas that we wanted to sketch and explore. So we continued with regular sketching at a slower pace, where we could refine the concepts and try out additional variations. This combination of fast Crazy 8 sketching and slower sketching afterward allowed us to continue developing ideas that either needed more time than the one minute limit or came up after the initial round.

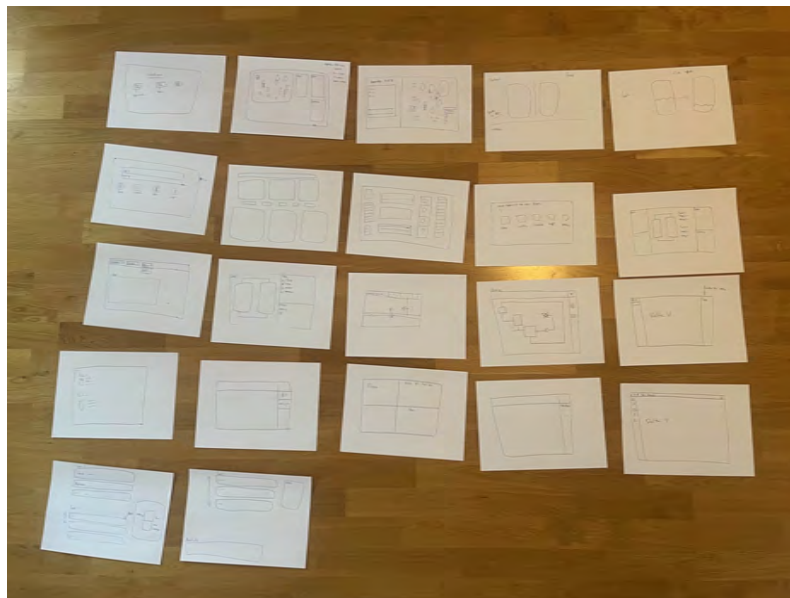


Figure 5.6: Overview of all sketched concept ideas laid out after the sketching session.

5.3.2 Voting Method

We used a sticker voting method to evaluate the concept ideas. Each of us received three green stickers and three blue stickers. Green stickers were used to mark ideas we saw as strong and worth keeping, while blue stickers indicated ideas where we liked specific parts. Before voting, we explained each idea to each other to make sure that we had a shared understanding. We then voted individually by placing our stickers on the sketches. After voting, we reviewed the distribution of stickers and discussed the results together to decide which concepts should move forward.

5.3.2.1 Voting results

The sticker voting showed a clear preference for one of the concept sketches. After individual voting, the results were analyzed by organizing the sketches into three

visual overviews: all sketches with their assigned votes, sketches that received green stickers, and sketches that received blue stickers. This made it easier to differentiate between concepts that were considered strong overall and those valued mainly for specific elements.

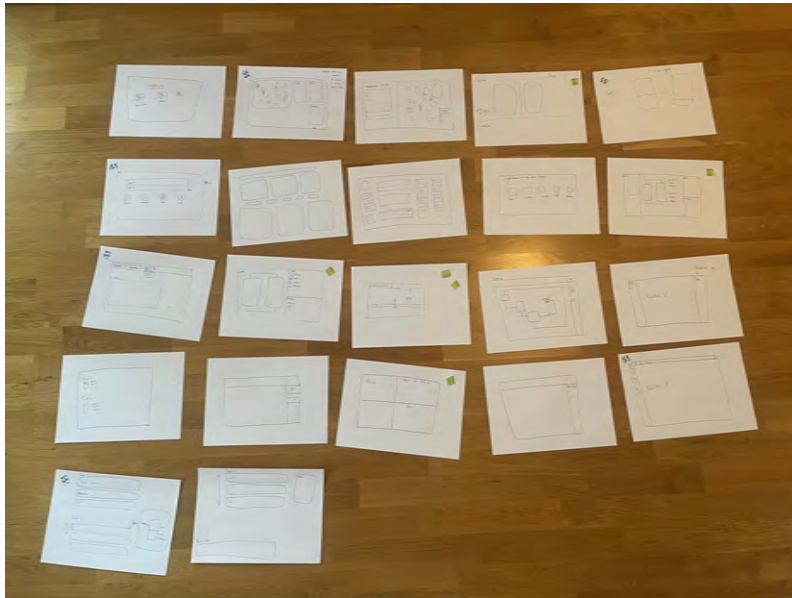


Figure 5.7: Overview of all sketched concept ideas laid out after the voting session and before the voting.

The sketch that was the only one to receive two green stickers was considered the strongest candidate and was therefore selected as the winning concept. During the followup discussion, this result remained unchanged, indicating a shared agreement within the group.

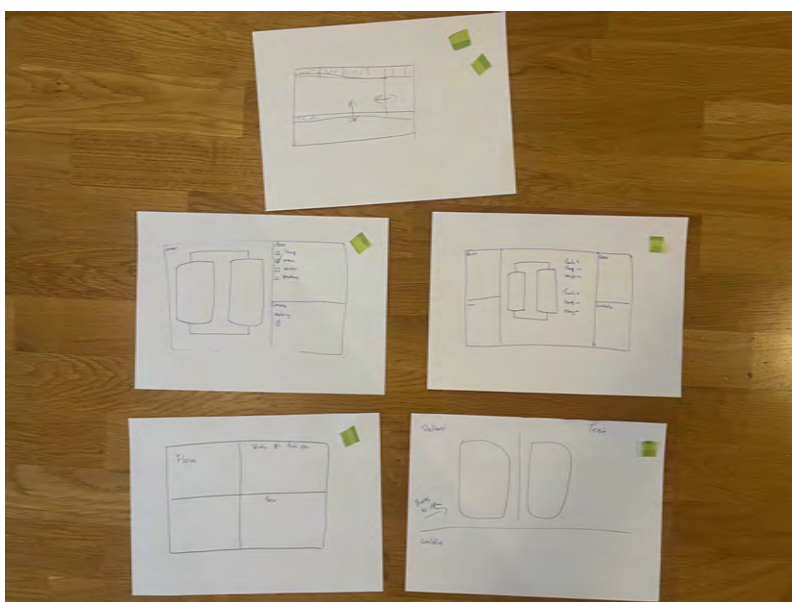


Figure 5.8: Concept ideas that received green stickers during the voting.

5.3.2.2 Key strengths of the selected concept

The winning sketch was mainly chosen because of its balance between familiarity and practical usability. The following aspects were highlighted as particularly strong:

- **Clear separation of areas**, making the interface easier to understand and navigate
- **Adjustable panel sizes**, allowing users to resize sections based on their current needs
- **A familiar layout**, where the most important content is given the most space and is presented first
- **Supporting information**, such as test results and error messages, placed to the side
- **A console located at the bottom**, reflecting typical interface layouts and user expectations

In general, the concept was perceived as intuitive and suited for its intended use case.

5.3.2.3 Feedback from blue stickers

The blue stickers highlighted several individual design elements that were considered valuable. These included the use of dropdown menus for simplifying selections, simple and clear visualizations of the tanks to effectively communicate changes in water and oil levels, and scrolling mechanisms to support file selection when handling larger sets of items.

However, some concerns were also raised. In particular, the results indicated the risk that the interface could easily become overloaded with features and information. This raised concerns that adding too many elements could reduce clarity and result in a visually cluttered design.

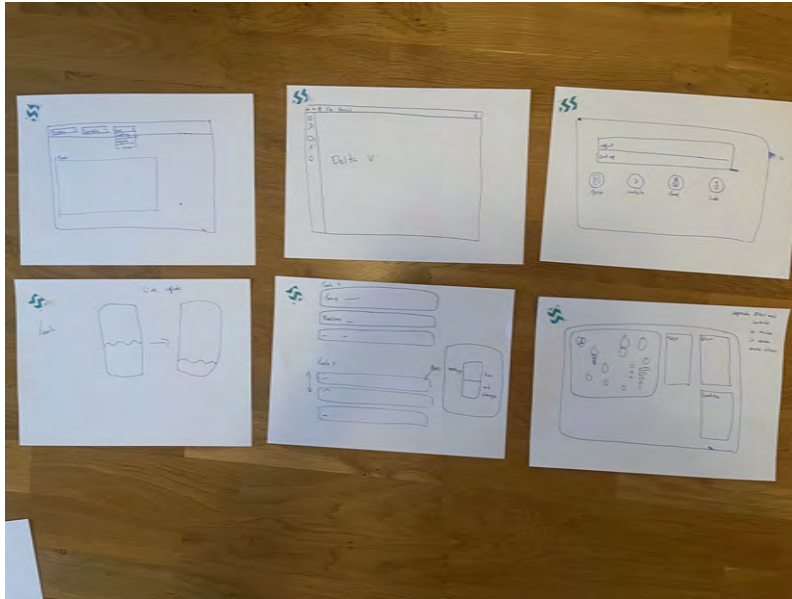


Figure 5.9: Concept ideas that received blue stickers during the voting.

5.3.2.4 Rejected concepts

The remaining sketches were not selected for further development. Some were considered visually appealing but lacked practical usability for the intended task, while others were perceived as too cluttered, making them harder to use and understand.

5.3.2.5 Outcome

Based on the voting results and the subsequent discussion, the sketch with two green votes was chosen as the foundation for further design work. Elements highlighted through blue stickers may be integrated where appropriate, while attention will be given to avoiding an overly dense or cluttered interface in the next design phase.

5.4 Requirements Gathering

Before the start of the implementation phase, a basic requirements specification was created to guide the initial discovery phase of the project. The requirements were based on the combination of the thematic analysis results, as well as previous developer domain knowledge and experience.

- **Process Control:** The system should let users execute both automated and manual operational sequences on the IPE.
- **Emergency Intervention:** The system must provide a mechanism to terminate/abort an active process or simulation upon user demand.
- **Conflict Prevention:** The system should prevent users from initiating simultaneous processes on the same hardware, leading to instability or failure.

- **Real-time Feedback:** The system should provide continuous updates on the current operational state and data from the connected equipment.
- **Physical State:** The system should provide a visual mapping of physical parameters, such as fluid levels, that corresponds to the actual state of the hardware.

5.5 Implementation

The primary objective of the implementation phase was to translate the selected design concept into a functional software prototype. This required establishing the backend architecture, implementing real-time data streaming, and developing the frontend interface.

5.5.1 Fading Scrum

During the implementation phase, we followed the Scrum framework, as previously mentioned. Initially, this provided the necessary structure to start developing the prototype in the form of a map forward. As the prototype was being developed and became more and more fleshed out, what is currently missing, and what to do next became more and more obvious. In other words, the project shifted from exploration to execution. We noticed that the formalities of Scrum, in particular the rigid cycles of sprint planning and backlog management started yielding diminishing returns in terms of keeping development on the right track. Since the immediate requirements became self-evident, we decided to transition to a flow-based approach to reduce administrative overhead. One key benefit with the Scrum framework is that it keeps developers in the loop, and "forces" communication among team members. Since the team in this project consists of only two developers, the outcome of this decision relies on our ability to maintain communication. We decided that since we are only two developers, the risk for a communicative breakdown is minimal. This let us put our focus on implementation velocity, while still maintaining the core iterative development with frequent internal checkpoints. The sprint reviews were replaced by ad-hoc demos, including testing, showing the other developer what has been implemented. Importantly, the code reviews remained a requirement for merging new code into the main branch. This way, we maintained our quality standards and ensured that no unreviewed code ended up where it does not belong. This transition also meant moving away from the traditional scrum backlog, we handled this by using a prioritized to-do list and new requirements and features were handled immediately, rather than moving through the scrum formality hoops.

5.5.2 Architectural and Design Impact of the Proposed Guidelines

Proposing the guidelines marked a transition point in the project, shifting the focus from user and context (the *define* phase) to implementation (the *develop* phase).

Throughout the implementation phase, these guidelines actively shaped the application, as well as both technical and design trade-offs in the developed prototype.

5.5.2.1 Presenting the System State

Guideline 1 (*Match the Physical Setup*) had significant impact on the GUI development trajectory. Initially, the data and measurements from the physical setup were presented in a tabular format, as the data is high-dimensional. However, guideline 1 shifted that design trajectory from a textual to spatial visualization. To align the interface with the physical state of the hardware, the frontend development revolved around rendering animated and dynamic components. Consequently, the implementation process shifted from basic data presentation to a high-fidelity visual layout designed to lower the cognitive load for the user.

5.5.2.2 Enabling Flexible Control

To satisfy Guideline 2 (*Flexible Test Control*), the backend architecture required a flexible design that can execute instructions from a pre-made script as well as letting the user execute instructions "on the fly", without needing to create a new script. This could be handled by executing the instruction directly, using the same library as the pre-made scripts. However, feedback from the stakeholder indicated that they would like the ability to create "custom scripts" using the GUI. Because of this, it was decided to implement custom instruction execution as creating a script based on a template, then executing that script as if it were a pre-made script. In essence, this let the script execution module to fulfill both requirements simultaneously.

5.5.2.3 Temporal Data Synchronization and Plot Stability

Guideline 3 (*Stable Views for Live Data*) dictated the state management strategies used during the development of the data view. Because the data feed is high-dimensional and continuous that cannot be paused at the hardware level, a method to isolate user exploration from incoming data ingestion was required. This led to a strict separation between the data fetching layer and the UI rendering loop. The backend continuously broadcasts telemetry data to the frontend, and rather than pausing the broadcasting, it was decided to cache the incoming data in the frontend during the pause, and immediately ingest it when resuming.

5.5.2.4 Logging, Persistence, and Error Handling

Following the 4th guideline (*Clear Error Handling*) could've been done in several ways. Messages could, for example, be displayed on modal panels/dialogs as pop-ups, or banners, status bars, or a dedicated log window. Modals/dialogs are best suited for critical messages such as errors, but unfitting for displaying general non-critical information. In addition, once the window is closed, the message disappears, making the contained information non-persistent. Instead, a dedicated log window was chosen as it lets the user access all types of messages, as well as the message history. Additionally, the log window does not block the rest of the application, giving the user the ability to navigate in the interface while troubleshooting an

error. Furthermore, the user can choose which message types should be saved in a separate log file, ensuring that messages are persistent through system crashes and restarts.

As the guideline mandated that messages should be easy to differentiate, the backend was refactored to categorize messages as either *INFO*, *OK*, *WARNING*, and *ERROR*. To further differentiate the messages, each category was assigned a color. Icons could have been chosen instead, but it was decided to use colors to avoid unnecessary visual clutter. Informal feedback highlighted that the color alone is enough to distinguish the message types.

5.6 Evaluation

1. Match the Physical Setup

Result: Pass

Observation: The interface implements a highly visual layout to represent numerical data, the animated water tanks and dynamic wave textures depict fluid movement and tank fill levels. By replacing numerical string decoding with spatial, metaphorical graphics, cognitive load is reduced for users, which addresses themes 2 and 4. The interface maps cleanly to the real world setup.

2. Flexible Test Control

Result: Partial pass

Observation: The prototype successfully implements a dual-mode interaction model, with the script execution environment alongside the "Manual Mode" control panel for simple/exploratory testing. However, while the script execution environment is restricted to only allow one script to run at a time, test sequences can still be executed from "Manual Mode", instead of being restricted or queued.

3. Stable Views for Live Data

Result: Pass

Observation: High-dimensional, time-dependent datasets are handled by a dedicated "Data View" featuring an interactive timeline with live plotting. The user is able to select which dimensions to display, and is able to browse the data by zooming and scrolling into specific periods to e.g inspect a strange ullage spike. When a specific period is selected, the feed is automatically paused, ensuring that the plot doesn't shift while the user is investigating. The system continues to stream and log incoming data packets in the background, but is plotted when the live plotting is resumed. This keeps users in control, while simultaneously ensuring no data is lost.

4. Clear Error Handling

Result: Partial pass

Observation: The backend system handles error as they occur, and delivers mes-

sages with a "severity level" ranging from ERROR, WARNING, OK, and INFO. Each severity level has its own color, with errors being red, warnings orange, OK green, and info blue. These messages are displayed in a console panel in the interface, along with their severity level, the user is also able to select which severity levels should be saved in a log-file for persistence. This means that system feedback is immediately recognizable, stored, and presented in a way that is easy to handle. However, error messages and warnings do not provide any actionable information, they simply state what is wrong without the cause.

Summary of the Evaluation

Overall, the evaluation shows that the interface is an effective tool for expert test engineers. It presents the real world hardware with up to date information in a way that is easy to interpret (Guideline 1). The interface provides both script execution capabilities, along with manual execution, but lacks validation to prevent conflicting concurrent inputs (Guideline 2). The time-series plot in the Data View locks the viewport on user interaction while caching live telemetry in the background, providing a stable view without data loss (Guideline 3). The console in the front-end displays messages from the system along with a severity level, providing visual hierarchy. This gives the user an overview of the messages in the console without having to actually read the messages and can quickly find what they are looking for (Guideline 4). Overall, the system handles complex testing well by remembering settings so the user doesn't have to, and it provides strict safety features like the "Force Stop" button. However, the evaluation also points out a missing safety feature in the form of preventing conflicting concurrent execution. Furthermore, since the software relies on raw, technical system logs, and doesn't actively block bad data entry, it is missing some of the basic safety nets and plain-English feedback you would expect in a modern UI. Ultimately, the system is very capable for those who know how to use it, and these findings give us a clear list of things to improve in the next version.

5.7 Refined Design Guidelines

The initial design guidelines, based on HCI theory and qualitative user interviews, provided a foundational framework for designing a prototype system that supports system verification. However, the evaluation of the prototype revealed functional gaps, edge cases, and systemic complexities that general usability heuristics do not fully account for in high-consequence contexts. The insights from the evaluation were used to refine the initial guidelines.

1. Match the Physical Setup

Insight: The evaluation validated that replacing raw numeric string decoding with a graphical layout successfully mitigated high cognitive load reported in themes 2 and 4. However, this guideline should go beyond static visual resemblances. The

interface must maintain high spatial fidelity, meaning that the visual position, directional animations, and dynamic states of graphical elements must explicitly mirror their physical counterparts.

Change: Rename to *Spatial Fidelity and Dynamic State Mapping*. The interface should maintain high spatial fidelity. Visual position, directional animations, and dynamic states of graphical elements must explicitly mirror their physical counterparts.

2. Flexible Test Control

Insight: While the implementation of a dual-mode environment (automated script execution alongside manual controls) addressed the workflow flexibility requested in themes 3 and 5, the evaluation exposed a vulnerability. The prototype fails to prevent concurrent command conflicts between automated scripts and manual commands. Additionally, adding the option for the user to "dry run" the specified command(s) would let them verify that the command does what they expect it to do.

Change: The interface should allow users to easily switch between running automated test scripts and making quick manual instructions without overriding each other. The user should also be able to preview the consequences of their configured instruction before executing. The guideline should further be refined to require user confirmation when executing commands.

3. Stable Views for Live Data

Insight: The evaluation confirmed the success of the "Data View" viewport behaviour. Locking the timeline and live-plotting visualization when necessary, determined by the type of interaction.

4. Clear Error Handling

Insight: The prototype's color-coded console panel and log-file persistence solved the visual clutter and data loss frustrations highlighted in theme 1. However, the evaluation revealed a secondary usability bottleneck as the error messages state what failed without explaining why or how to recover.

Change: In addition to providing a clear distinction between different message types, and storing them for persistence, the messages should be actionable. In other words, the message should not only say *what* went wrong, but also *why*.

5. Proactive Input Validation and Defensive Safety Nets

Insight: A major finding from the evaluation summary was that the system lacked preventative barriers, relying heavily on the expert knowledge of the user to avoid inputting bad data. This gap is present in the initial guidelines, and should in hindsight, be a guideline.

New guideline: The interface must act as a proactive filter at the presentation layer. It must reject invalid data formats or values that exceed safe operational threshold, visually flag the boundary violation, and force safe defaults before any command is executed.

6. Data Filtering

Insight: While we implemented functionality that lets the user select the data streams that they want to display, the guidelines did not capture this aspect of live data plotting. Therefore, an additional guideline is emergent. **New guideline:** The interface must provide filtering mechanisms that allow users to isolate telemetry streams by subsystem, and type. It must support the creation of saved filter presets to avoid forcing the user to recreate commonly used filters, reducing cognitive load without compromising underlying log persistence.

6

Results

The results aim to answer the research question "*What should be considered when designing interactive simulation systems for industrial equipment to support system verification through interactive information visualization?*". The answer to the research question consists of a set of proposed guidelines for designing said systems. The proposed guidelines were used when developing Viper.

6.1 Viper

The development process resulted in a web-based application that facilitates user interaction with the IPE. The app consists of a frontend as well as a backend. The Viper application lets the user execute test scripts, monitor the IPE in real time as well as visually browse collected data. Additionally, Viper features a "manual mode" that lets the user execute the fundamental operations without needing a script. The interface is split in three parts; a main view, a "test tools" panel, and a console panel.

The three-panel layout was chosen over a multi-page structure, where each feature would have its own dedicated page to reduce navigational complexity. Keeping the primary tools visible and located on a single screen aligns with the interaction design principle of minimizing the user's need to remember where functionality is located across different views[18]. Engineers who tested the interface reported that it felt intuitive and that they could easily build a cognitive map of where to find what they were looking for. This finding is consistent with the IxD goal of supporting efficient, low-friction interaction[16].

6.1.1 System Overview

The system overview consists of a visual representation of each tank that the system is connected to. The visual representation shows the current level in the tank, in addition, there are pipes that show how the liquid is moving. Each tank also has a drop-down menu that contains the available information about the tank, such as level and flow measurements, the target level, and pump rate divided into categories respectively. This provides numerical readouts on demand.

A purely numerical display was considered but rejected in favour of the graphical representation. According to the principles of cognitive and perceptual support in

information visualization, visual encodings should leverage natural associations to reduce interpretive effort[59]. A filling tank graphic maps directly onto the physical reality the engineers work with, utilizing what the literature describes as a natural or metaphorical association; high fill level corresponds to a visually filled tank shape. This reduces the cognitive work required to assess the system state at a glance and immediately shows the level in relation to the tank capacity. The numerical readouts are retained but placed behind a drop-down, following the details-on-demand principle of presenting summary information first and allowing users to access specifics when needed[26]. This also supports the visual representations dimension of IxD, by using graphical elements that aid comprehension rather than requiring the user to interpret raw numbers under time pressure[16].



Figure 6.1: A screenshot of system overview

6.1.2 Data View

The data view plots incoming data in real time, giving the user the ability to see exactly what has happened in the system. The user is able to choose the displayed measurements and the axis' scales will adjust accordingly. The plot allows the user to select an area with a click-and-drag input, which will pause the plot and zoom in on the selected timespan. By double-clicking, the plot zooms out and resumes. During the pause, data is still collected, and the data collected during the pause is amended, meaning that no data is lost by pausing. By using the scroll-wheel, the user can adjust the x-axis resolution.

The interaction model was designed around the principle that the user should always remain in control of the view, consistent with the IxD dimension of time. Media that changes over time should not shift against the user's intent[16]. A horizontal scrollbar was considered as a navigation mechanism, but click-and-drag selection was chosen because it simultaneously selects and zooms, reducing the number of interactions required to inspect a specific timespan. The filtering capability, toggling which datastreams are visible, follows the approach demonstrated by Wang et al.[26],

where users construct a customized view by including or excluding datastreams rather than being presented with everything at once. This is particularly relevant given the large number of datastreams the system handles.

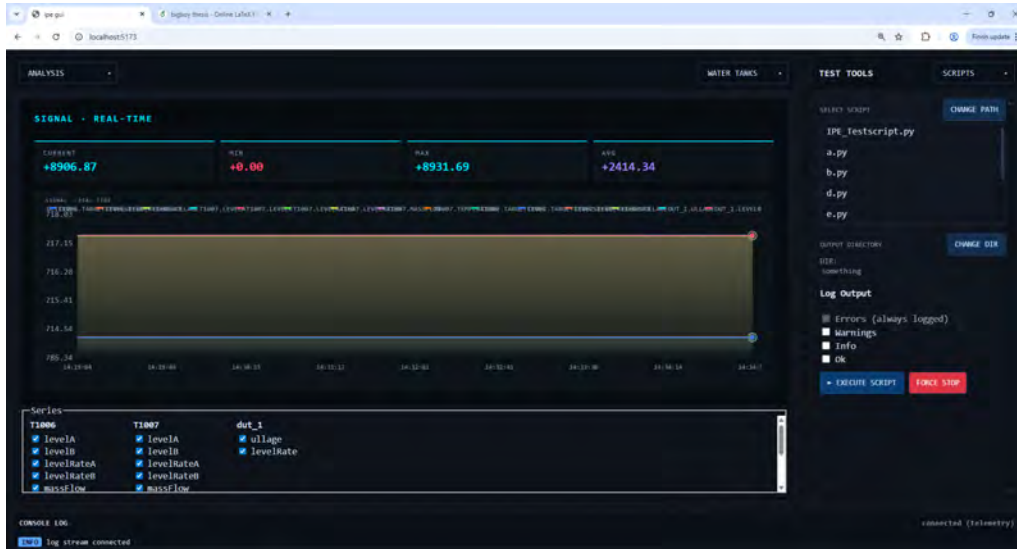


Figure 6.2: A screenshot of Data View

6.1.3 Test Tools

The Test Tools panel contains two views, *Scripts* for running existing scripts and *Manual Mode* for manual execution.

6.1.3.1 Scripts

The Scripts view contains a list of available scripts to run. Should the desired script not be located in the configured script directory, the user can change the script directory path. The user is also able to configure the output directory path, so the system knows where to put the log for that run. Before executing the script, the user is also able to choose which log message categories that are saved in the logfile. Finally, the execute button is available if a test is currently not running, and the execute button starts the test. While a test is running, the execute button is disabled, and a bright red "Force Stop" button is enabled. This button aborts the test that is in progress.

The button swap is an application of the Words dimension of IxD[16], the label communicates not just an action but the current system state and the severity of the available action. The red color follows the established color convention used throughout the interface where red is reserved exclusively for critical actions. This reduces ambiguity through consistency[18].

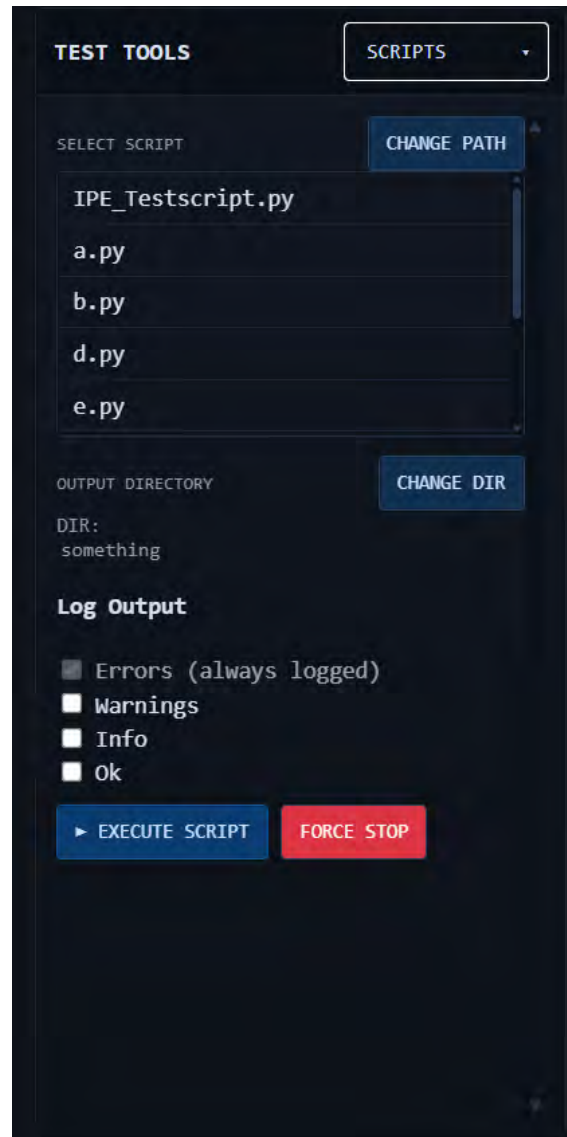


Figure 6.3: A screenshot of the part to handle Scripts

6.1.3.2 Manual Mode

The Manual Mode view lets the user configure an operation and select the tank on which it should be executed on. After that, the user can either run it directly, or save it as a script that can be executed later via the *Scripts* tool. This is meant to be an intuitive and easy way to execute simple operations without needing to write a whole script. When the *Run* or *Save* button is clicked, the input is collected and validated, and the backend uses a Jinja2 template to create a script that is then executed.

Manual Mode addresses a specific usability concern: requiring engineers to write scripts for simple, one-off operations introduces unnecessary friction for tasks that do not warrant a full script. By providing a structured form interface, the design reduces the expertise required to perform basic operations while still integrating with the existing script infrastructure. The save-as-script option further supports

learnability[18]. Users can inspect the generated script to understand the underlying format, gradually building competence with the scripting system without being required to start from scratch.

TEST TOOLS MANUAL MODE ▾

Tank: **T1009** ▾

TANK SETUP

Empty:

Full:

MOTION SETUP

Mode: **STEPPING** ▾

PARAMETERS

Stepsize:

Velocity:

Cycles:

OUTPUT DIRECTORY CHANGE DIR

DIR: something

NEW LOAD RUN SAVE

Figure 6.4: A screenshot of the part to handle Manual mode

6.1.4 Backend

6.1.4.1 Overview

The backend is implemented using the FastAPI framework because a core requirement of this system is handling multiple concurrent WebSocket connections while simultaneously running background telemetry polling tasks. FastAPI runs on Uvicorn, an ASGI server, meaning concurrency is handled via Python's *asyncio* event loop rather than threads or processes. This makes it easy to implement tank and dut telemetry tasks as async background coroutines. A WSGI framework like Flask

would require additional components such as threads or a task queue to achieve the same functionality. Additionally, request and response bodies are declared as Pydantic models, which means that type validation, coercion, and error reporting happen automatically at the boundary of the application. This is well-fitting for script execution requests, where improper input should be rejected before reaching business logic.

The backend architecture consists of several layers: *routers* $\rightarrow$ *managers* $\rightarrow$ *services*. For control operations, REST (HTTP) is used as the communication protocol, and for real-time streaming, WebSockets are used instead.

6.1.4.2 Entry Point and Configuration

The Viper application follows a factory pattern, where main is deliberately minimal, doing nothing more than calling the *create_app()* function and exposing the result as the ASGI application for Uvicorn. FastAPI's *lifespan* context manager handles initialisation. On startup, telemetry sources are instantiated and their respective background tasks are launched as asyncio tasks. The tasks run concurrently for the lifetime of the application without blocking the main event loop. The *ConnectionManager*, script directory path, and output (log) directory path are attached to the app state, making them accessible to routers without dependency injection or global variables. For CORS (Cross-Origin-Resource-Sharing), only *localhost* : 5173 is allowed, in other words, only the host is allowed. There are three registered routers, *scripts*, *tanks*, and *websockets*, corresponding to the three domains of the API.

6.1.4.3 API Layer

The API layer features REST endpoints for script control and WebSocket endpoint registration for telemetry and log streams. Request validation is done via Pydantic models.

6.1.4.4 Real-Time Communication: ConnectionManager

The real-time communication layer is built on WebSockets, managed by a central ConnectionManager and three dedicated router endpoints. Together they handle live telemetry streaming and log delivery to connected clients. The ConnectionManager owns a set for each type of client, as well as the broadcast logic and log replay buffer. Client registration and deregistration is handled directly in the router endpoints, this is a pragmatic design that keeps the manager focused on broadcasting concerns while delegating connection lifecycle to the router layer.

The WebSocket endpoints serve distinct subscriber groups and all of them follow the same structural pattern: accept the connection, register the client in the corresponding set. The endpoints are effectively unidirectional, the server pushes data continuously, and the only meaningful client action is closing the connection.

The broadcasting mechanism consists of an internal method that serialises the message to JSON, then sends it to all clients in the target set. Failed sends result in the client being declared dead and pruned from the set after the broadcast completes.

Log messages originate from script execution and are routed to connected clients via a factory method that returns an async callable used by the script executor. Each call to this function writes the log entry to a RunLogger instance for file persistence, then broadcasts it to all log clients. This design keeps the script execution decoupled from WebSocket concerns, the executor simply receives an async function and does not need to care about the connection manager or WebSocket logic.

All log entries are structured as JSON objects containing a severity level, message string, and a timestamp, such that they can be meaningfully correlated with telemetry data.

6.1.4.5 Telemetry Streaming

Each telemetry type has a corresponding task running concurrently with a polling interval of 0.5s, this was arbitrarily chosen as a middle-ground between performance and accuracy. The message schema is a timestamp + readings broadcasted as JSON. The test facility stream is separated from the DUT (Device-Under-Test) because they come from different sources.

6.1.4.6 Service Layer

The service layer serves as an abstraction between the application and external data sources, the services ensure that all telemetry messages follow the same format, regardless of where it came from. This also means that the same telemetry WebSocket works for all telemetry sources. The service layer uses existing libraries to communicate with external systems, in this case Lemur and DeltaV.

6.1.4.7 Script Management

Scripts are stored as plain Python files in a directory configured at application startup, this can however be changed by the user through the frontend. Script execution requests are modelled using Pydantic, which enforces several constraints at the API boundary before any execution logic is reached. The constraints include checking the file extension, only allowing .py files. A reasonable timeout duration is enforced, preventing indefinitely running or instant executions. Parameter keys are validated to be alphanumeric (with underscores), mitigating shell injection risk through parameter names. However, the risk for shell injection is not a concern as this application is accessible only to trusted users.

These constraints are enforced by Pydantic validators and result in a 422 response with descriptive errors if violated, without any custom error handling code in the router.

Before execution, the requested script name is resolved to an absolute path, that is then checked against the scripts root directory, ensuring that path traversal attempts are caught and rejected. However, there is a script path override field that bypasses this, allowing execution of any .py file in the system, and would be a security consideration if the system is ever exposed beyond the trusted local network.

Scripts are executed as child processes subprocess and asyncio to avoid blocking the event loop during process creation. Parameters from the payload are passed as command-line arguments. The Python interpreter used is resolved from the PYTHON environment variable, with a fallback to the same interpreter running the FastAPI application removing virtualenv configuration requirements. Stdout and stderr are captured as pipes and streamed concurrently, each line read from either pipe is forwarded as a log entry with stdout at INFO level and stderr at ERR level via an injected log function. This lets script output appear in the frontend log view in real time as the script runs, rather than being delivered as a batch on completion.

On process completion, a return code is retrieved via a blocking call, but since the process is finished, the blocking duration is negligible, but should be non-blocking. The return code determines whether the final log entry is a OK message on success or ERR with the exit code on failure or early exit.

The log pipeline integration works by injecting a log function at ScriptExecutor construction time, meaning tha the ScriptExecutor has no direct knowledge of WebSockets, the ConnectionManager or file I/O. All log output flows through this single function, which handles both WebSocket broadcasts and file persistence. This decoupling means the executor is independently testable, and does not need the whole application in the environment.

6.1.4.8 Design Decisions and Trade-offs

The API combines two communication paradigms serving different purposes. Control operations, like listing scripts, initiating execution, and termination, are handled via stateless HTTP endpoints. These interactions are request-response by nature, the client sends a request and receives confirmation, and the interaction is complete. Real-time data delivery is handled via persistent WebSocket connections. This design maps each paradigm to the interaction pattern that is most suitable, rather than forcing all communication through a single mechanism. Polling a HTTP API for telemetry and logs would introduce unnecessary latency and request overhead for data that is continuous.

The system enforces a constraint that only one script may execute at a time, this significantly simplifies the concurrency model. Since only one script runs at a time, there is no need to manage a job queue, track multiple concurrent processes, or arbitrate access to shared hardware resources. Given this constraint, the built-in asyncio event loop is used.

Using a task queue like Celery would require a lot of additional infrastructure like a message broker, worker processes, and result backend, which do not provide any real benefit for a single-consumer execution model. It would however be the appropriate choice if the system was designed to support queued or concurrent script execution, but is overly complicated for the current requirements.

Dead connection handling works by detecting disconnected WebSocket clients during the broadcast cycle, if the send fails, the client is collected into a local set and removed from the subscriber set after the current broadcast completes. Worth noting

here is that a dead connection is not detected until the next broadcast attempt, meaning it persists in the client set for up to 0.5 seconds in the case of telemetry clients. Secondly, a client that has been disconnected but not yet removed from the subscriber set will receive one additional failed send attempt. Given the polling frequency and the non-critical nature of the monitoring interface this is acceptable behaviour. Using a heartbeat-based approach instead would detect dead connections proactively but would add significant complexity and additional async tasks without meaningful benefit at this scale.

The backend serves the compiled frontend assets directly via a FastAPI mount rather than through a dedicated web server, this simplifies deployment to a single process with no additional infrastructure, which is appropriate for a test facility application running on a local network. However, under high concurrent client load this would become a bottleneck but is considered out of scope for this application.

6.2 Proposed Guidelines

We propose that in addition to traditional, well-established guidelines such as Shneiderman’s golden rules[57] or Nielsen’s usability heuristics[56], the following six guidelines should also be considered when designing an interface for a highly specialized system verification context.

1. Spatial Fidelity and Dynamic State Mapping

Guideline: The interface should maintain high spatial fidelity, meaning that visual positions, directional animations, and dynamic states of graphical elements must explicitly mirror their physical counterparts. In other words, what the user sees on the screen should be easily translated to the physical world. Figure 6.5 shows how Viper displays two tanks that are connected via a pipe. The arrows indicate the flow direction, and the tank graphics update according to their fill level.



Figure 6.5: Example of spatial fidelity

2. Flexible Test Control

The interface should allow users to easily switch between running automated test scripts and making quick manual instructions without overriding each other. The user should also be able to preview the outcome of their configured instruction before committing to execution. The interface should also ask for user confirmation to prevent mistakes.

Figure 6.6 shows the different control modes the user can choose between in the Viper GUI. The user can choose their desired "test tool", depending on what best suits their current objective.

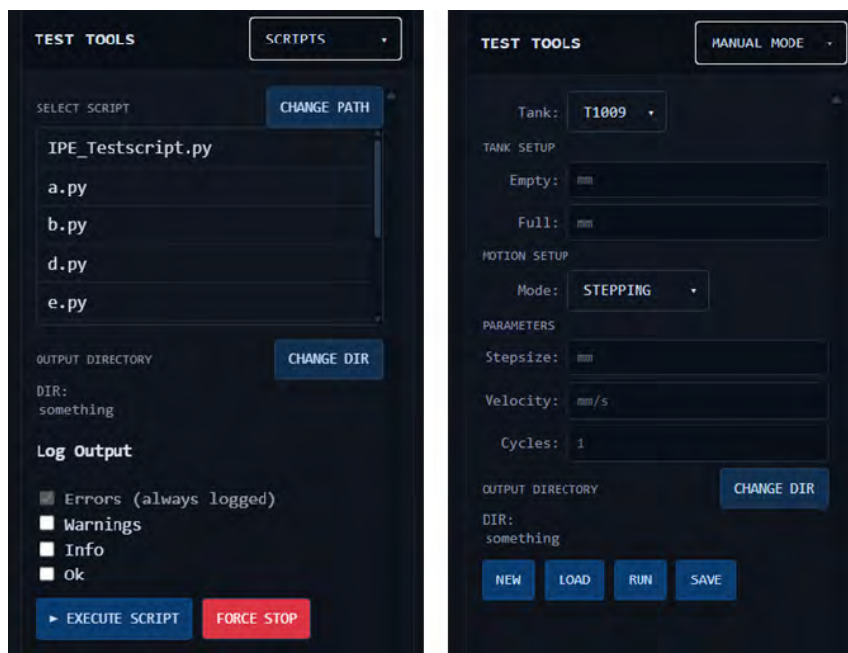


Figure 6.6: Example of flexible control modes

3. Stable Views for Live Data

The interface must never auto scroll or force view changes when the user expects it to remain static, e.g when the user is trying to analyse a specific event in a live data stream. Figure 6.7 shows what the frozen view might look like, with an indicator telling the user that the feed is paused, along with a simple instruction for how to resume.

4. Clear Error Handling

The interface should provide a clear distinction between different types of system feedback, ensuring that errors are immediately recognizable and stored in a way that is easy for engineers to handle. As shown in fig. 6.8, the message consists of two parts, type and content. This tells the user what type of message they are reading before they read the content, this helps the user quickly identify relevant messages. The messages are then stored in same format for persistence.



Figure 6.7: Example of paused live data view

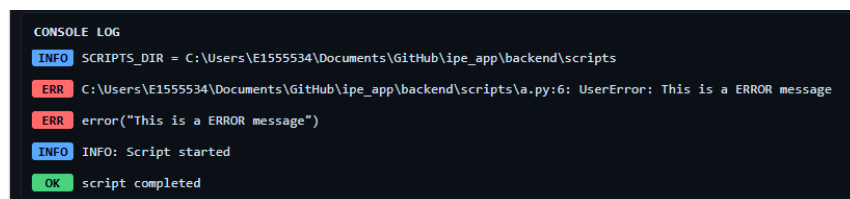


Figure 6.8: Example of distinction between types of system messages

5. Proactive Input Validation and Defensive Safety Nets

The interface must act as a proactive filter at the presentation layer. It must reject invalid data formats or values that exceed safe operational thresholds, and visually flag the boundary violation. The interface should also present the threshold and/or the valid data format, depending on why the input was rejected.

6. Data Filtering

The interface must provide filtering mechanisms that allow users to isolate telemetry streams by subsystem, and type. It must support the creation of saved filter presets to avoid forcing the user to recreate commonly used filters, reducing cognitive load without compromising underlying log persistence.

Viper handles this by simply letting the user select which data streams they want to display, as shown in fig. 6.9. Similar methods are used by Zhao et al. for durability test data[8].



Figure 6.9: Data filtering

7

Discussion

7.1 Supporting System Verification Through Visualization

The core purpose of Viper is to support engineers in verifying the behavior of level measuring equipment by making the system state visible, interpretable, and interactive. The evaluation results suggest that the interface largely achieves this goal within its intended context of use.

The graphical tank visualization directly addresses the challenge of communicating continuously changing physical states to users during operation. By encoding fill level as the height of a liquid column rather than a numerical readout, the design leverages what the information visualization literature describes as natural and metaphorical association[59]. A visually tall column intuitively corresponds to a high fill level without requiring the user to interpret or mentally convert a number. This is consistent with Cleveland and McGill’s finding that positional encodings on a common scale are decoded most accurately by the human visual system[24], and supports rapid status assessment in a monitoring context where engineers may want to take in system state at a glance. The availability of numerical readouts on demand reflects the details-on-demand principle[26], ensuring that precision is accessible without cluttering the primary view. Additionally, fluid flow is encoded as a wave moving in the direction of flow, leveraging the same metaphorical association. By visualizing flow, the user is not required to infer it from looking at the changing liquid columns.

The real-time data view similarly applies established InfoVis principles. The ability to filter data streams, zoom into specific timespans, and scroll through historical data allows engineers to construct a customized view of the system’s behavior at the level of detail relevant to a specific verification task. This approach is consistent with the visual analysis methods described by Zhao et al.[8], where separating a test process view from detailed channel data enables users to move between overview and inspection without losing context. The informal feedback from engineers supports this: users reported that the interface felt intuitive and that they could quickly build a cognitive map of the system, an outcome that aligns with the IxD goal of supporting efficient, low-friction interaction and reflects well-established usability criteria of learnability and memorability[18].

7.1.1 Interpreting the Evaluation Findings

The heuristic evaluation based on our initial four custom guidelines resulted in two passes and two partial passes, suggesting a generally effective interface with specific, addressable weaknesses. It is worth interpreting these findings in terms of what they reveal about the design priorities and trade-offs made during development.

The successful fulfillment of both Guideline 1 (Match the Physical Setup) and Guideline 3 (Stable Views for Live Data) demonstrates that the design effectively supports the operational workflow of the test engineers. The integration of domain-specific terminology such as "ullage," the implementation of visual tank models, and the preservation of viewport stability during live data streams collectively indicate that the system was shaped by a clear understanding of the target environment. This is consistent with the ISO 9241-11 definition of usability as inherently context-dependent[17]. Design decisions that may be inappropriate in a general consumer application are well-suited to technically proficient users operating within a highly specialized context.

The two partial passes deserve closer reflection, as they expose gaps between the system's capabilities and its communication of those capabilities to the user. The partial pass for Guideline 2 (Flexible Test Control) occurred because while the system successfully provides a dual-mode environment for both script and manual execution, it lacks the validation necessary to prevent concurrent command conflicts. The partial pass for Guideline 4 (Clear Error Handling) showed that while errors are visually categorized by severity, the messages only state what failed without explaining why or how to recover. Detailed diagnostic information exists, but it is presented in raw technical form, requiring significant cognitive effort from the user to interpret it.

Together, these findings showed that the initial guidelines were insufficient for the strict safety demands of a live industrial environment. The evaluation exposed a core vulnerability: the system placed the burden of catching mistakes entirely on the user rather than the interface itself. Entering the wrong parameters or running conflicting commands could easily produce invalid test results, or worse, risk hardware instability. This realization directly led to the refinement of the framework in Section 5.10, most notably resulting in a completely new, fifth guideline: *Proactive Input Validation and Defensive Safety Nets*. By iterating on the evaluation results, we ensured that the final guidelines require the interface to act as an active filter, rather than relying solely on the user's expertise to avoid making critical errors.

7.2 Context of Use and Design Trade-offs

Several design decisions that might appear as weaknesses in a general context are better understood as deliberate trade-offs given that specific deployment environment and specific user group. The system is designed exclusively for use on a trusted local network by a small group of expert engineers. This context justifies decisions such as the script path override that bypasses directory restrictions, and the presentation of

raw log output in the console, which would be inappropriate for a general audience but is informative and expected for the target users.

This reflects a broader principle in usability: as the ISO 9241-11 standard emphasizes, effectiveness, efficiency, and satisfaction must be evaluated relative to specified users, goals, and contexts[17]. The system's design is well-matched to its context in many respects, and several apparent shortcomings dissolve when viewed through this lens. The wicked problem framing[30] is also relevant here. The design challenge involved trade-offs with no objectively correct resolution, where solutions could only be evaluated as better or worse, rather than right or wrong. The decision to prioritize operational transparency over guided error handling is one such trade-off that is reasonable given the user group, but worth revisiting as the system grows or its user base expands.

7.3 Limitations of the evaluation

The evaluation methodology has limitations that should be acknowledged explicitly. The heuristic evaluation was conducted by the developers of the system, which introduces a risk of confirmation bias as evaluators who designed the interface may unconsciously interpret ambiguous cases in its favor, or fail to notice violations that have become invisible through familiarity. To mitigate this effect, Nielsen's heuristic evaluation methodology recommends the use of three to five independent evaluators[18].

Furthermore, heuristic evaluation is an expert inspection method that predicts likely usability problems based on established principles, but does not measure actual user performance or experience. The informal feedback is encouraging and directionally consistent with the evaluation findings, but it does not constitute structured user research. A think-aloud study or task-based usability test with the users would provide qualitative evidence, rather than asking "does the interface conform to guidelines?". The qualitative evidence from a think-aloud or task-based study would show whether the users can efficiently and accurately complete their tasks using it. This is the most significant gap in the current evaluation, and is caused by time constraints.

7.4 Reflections over the Process

Overall, the working process throughout the thesis was successful, and we ultimately achieved the goals set out in our initial planning. However, the final phases could have been smoother. The later stages of implementation became somewhat stressful as the project drew to a close, requiring a heavy push to finalize the code. Despite this end-of-project crunch, the core of our workflow was solid and kept us on track.

7.4.1 Evolution of the Workflow

Starting the project with a strict Scrum framework was something necessary. In the beginning, the problem space felt open, and we honestly were not entirely sure what

we were going to implement. Working in two week sprints provided stability and forced us to produce results instead of getting stuck in endless research. Additionally, using a Trello board made our progress impossible to ignore. We could easily see our progress, and if we experienced a loss of momentum, the board made it obvious right away, helping us get back on track.

Another practice that worked surprisingly well was the Sprint Retrospectives. Even though we had worked together on group project in previous university courses, conducting a 20-week thesis with just the two of us in a corporate environment was a different experience. Setting up a dedicated time to openly discuss our sprint made it much easier to have difficult conversations. It relieved the pressure of bringing up personal stress, bad habits, or workflow issues because it became part of our routine.

Looking back, our initial Agile setup acted as a set of guardrails. It kept us moving in the right direction when things were uncertain. Once we moved deeper into the implementation phase and had a solid grip on the project, we matured enough as a team to essentially manage our own pace. We no longer needed the strict rules, which is why dropping the formal meetings and transitioning into a more fluid, continuous workflow felt natural.

7.4.2 Executing the Design Process

When it comes to the design process, moving from the pre-study into the interviews, the thematic analysis and the sketching and voting, things went largely as expected. We had practiced this sequence many times throughout our Master's program that the workflow felt natural.

The only real obstacle during the project happened during data collection. One of the participants refused to sign the consent form. Losing a participant when operating with a small sample size is frustrating, but there was not much we could do about it. We followed standard procedure, dropped their data from the analysis and move on. In university projects, people usually cooperate easily, but this served as a good reminder that in a real corporate environment, you cannot always control the human factor. You simply have to handle it ethically and keep the project moving.

7.4.3 Technical Implementation and the Impact of Guidelines

Reflecting on the technical implementation, we were fully aware from the start that the core of this project would center around frontend development. However, because our primary programming strengths lie in backend architecture, translating our design concepts into a functional interface required a period of adjustment. Developing the dynamic visual features was not a fundamental technical barrier, but rather a learning process that required us to invest extra time to master the necessary frontend technologies. Since the frontend was the primary focus of the thesis, this learning process did not derail the project, but it did make the initial development phase more time-consuming as those extra hours naturally added up.

Working directly with real industrial hardware also had a noticeable impact on our development speed. Rather than building a completely independent application from scratch, we had to navigate existing corporate technical guidelines, integrate seamlessly with legacy software, and respect strict operational safety boundaries. While no single constraint was overwhelmingly difficult, ensuring our code functioned safely alongside real-world hardware meant that individual tasks naturally took slightly longer, and these minor time investments compounded over the course of the project.

The implementation process was also heavily influenced by our custom design guidelines, which ultimately acted as a double-edged sword. On one hand, they were highly beneficial in providing a clear roadmap. They dictated our focus, helped us prioritize critical features, and kept the development process anchored to the users' actual needs. On the other hand, because the initial problem space was so broad and undefined, strictly adhering to these guidelines may have prematurely narrowed our focus. By heavily prioritizing the specific features dictated by our framework, we inadvertently constrained our own creativity, potentially leaving alternative, innovative design solutions unexplored.

7.4.4 Areas of improvement

If we had to do the project all over again, the biggest difference we would make is in how we tested the software during the development. While we did do frequent, informal check-ins with the test engineers and our mentor, we never sat down to do any official usability testing with them. In hindsight, we should have worked in more structured iterations. Giving the users early, functional versions of the prototype to test would have allowed us to refine the design much earlier in the process. Working this way would have also helped us manage our time better by pushing us to hit specific testing milestones, potentially leading to an interface that was even better suited for the end users.

The second thing we would change is in the pre-study phase. We spend a lot of time having the engineers explain and demonstrate the tools they currently use. While those walk-throughs were helpful, watching someone else use a complex system is never the same as using it yourself. If we were to do it again, we would definitely spend more time getting hands-on with their existing software. Gaining that practical experience would have given us a much clearer and more realistic understanding of the exact problems the engineers face every day.

7.4.5 Recommendations for Future Projects

If we were to give advice to someone undertaking this exact same type of project, designing an interface for industrial testing equipment. Our first recommendation would be to immerse yourself in the engineers' actual workflow immediately. Do not only rely on them explaining the system. Go sit with them while they actually run a live test. Watch how they interact with the data streams, see what happens when an error pops up in real time, and get hands-on with the scripts and software they

currently use. Understanding the reality of their daily troubleshooting and analysis process makes designing the new digital tools significantly easier.

Another thing is remembering who your users are and understand what they need. It is always nice to have something that is simplified and easy to use, but it is also important to remember that test engineers are domain experts who need efficiency, control, and powerful data analysis. Our advice is to spend less time worrying about aesthetic minimalism and more time perfecting how the system handles and visualizes massive amounts of live telemetry. Features that allow the user to easily filter, zoom, and navigate through complex data without losing the live feed are what will actually bring value to their daily workflow.

7.5 Validity and Generalizability

To rigorously assess the academic and practical contributions of this result, the findings must be evaluated through the lenses of validity and generalizability. As a RtD project conducted within a specialized context, specific methodological constraints inevitably impacted the scope and certainty of the outcomes.

7.5.1 Methodological Validity and Evaluation Bias

A primary constraint on the internal validity of this study lies in the evaluation phase of the prototype. Due to strict development timelines, it was not possible to conduct formal, structured usability testing. It would have been highly preferable to conduct a think-aloud study or task-based study. A consequence of this is that the final evaluation relies solely on a developer-led heuristic assessment based on the proposed guidelines.

A significant flaw with this approach is the inherent risk of confirmation bias, as the creators of the prototype also assess its compliance with the proposed guidelines. This risk was mitigated by anchoring the evaluation directly in established HCI frameworks, specifically Nielsen's heuristics and Shneiderman's rules of interface design. Furthermore, the continuous collaboration with the stakeholder throughout the design process ensured that the design remained aligned with their demands and needs, preserving qualitative validity of the design choices.

A further consideration is the participant sample size. While five domain experts were initially engaged, the final dataset used in the thematic analysis had a sample size of $N = 4$. In standard consumer-facing software development, a sample size this small would severely threaten construct validity. However, because of the highly specialized context, and the fact that the interface is targeted specifically towards that user group, this is seen as acceptable. Given Nielsen and Landauers 5 user guideline, the sample size is not necessarily a problem, but we should have conducted more rounds where the domain experts were involved.

7.5.2 Generalizability of the Guidelines

The generalizability of the findings relates to whether the proposed design guidelines can be successfully extrapolated beyond the specific use case in this project. On a surface level, certain implementations within our prototype are highly domain-specific. The graphic metaphors mandated by Guideline 1, such as rendering dynamic water tank levels and directional wave-piping are tightly coupled with radar level measurement infrastructure and would not directly translate to unrelated industrial equipment.

However, the underlying structural principles defined by the six proposed guidelines possess a high degree of generalizability for the broader field of Interaction Design. The core concepts, such as decoupling the data ingestion layer from the UI rendering loop to maintain a stable view during live data streams (Guideline 3), and constructing flexible controls that support several alternative workflows, including both manual and automated operation (Guideline 2) address fundamental challenges inherent to complex data and control systems. Consequently, these guidelines can be generalized to the design of interfaces for any industrial equipment characterized by high-dimensional and time-dependent telemetry.

7.6 Ethical and Social Considerations

When looking at the social impact of this project, our main focus was on how the tool affects the engineers' daily work. The goal of automating the test scripts and making the data easier to read was never to replace the engineers or automate their jobs away. Instead, it was all about improving their work environment. By getting rid of frustrating, repetitive configurations, the system is meant to reduce their cognitive load so they can focus on actual problem solving rather than manual setup.

When it comes to research ethics, we made sure to keep everything transparent. All interview notes were kept anonymous and everyone knew exactly what the project was about. As we mentioned earlier, when one participant did not want to sign the connect form, we fully respected that and dropped their data from the study. It was important that the participation was always completely voluntary and handled properly.

When it comes to the ethical side of accessibility. We started by building the interface with a dark theme simply because we both personally use dark mode on our own devices and felt it looked better. We did think about making a regular light theme as well, but we recognized early on that it was not our main priority. Our main focus was to get the core functionality to work, so we made the decision that building multiple themes was outside the scope of this project. Along with the dark theme, we used colors like red and green to show different system states. Even though we do not rely only on color since there is text and other information, the color coding is still a massive part of how the interface communicates warnings versus normal data. Since full accessibility was not the primary focus for this prototype, building a

light theme and adding clearer visual cues is definitely an important step for future work.

7.7 Future work

Even though the current prototype successfully answers our research question and provides a solid foundation, there are several clear steps for how the project could be developed further.

As we mentioned in the ethical considerations, the most immediate design improvement would be addressing the accessibility issues. The next step for the interface would be building a light theme alternative and adding distinct visual icons or patterns to go along with the status colors. While this would not automatically make the entire system perfectly accessible to everyone, it would be a step toward making it much more inclusive. It would make the system significantly safer and easier for users with color blindness to use without relying heavily on colors.

Another major focus for the future development would be expanding the information Visualization tools. The core purpose of InfoVis is to amplify cognition and allow users to discover relationships through visual thinking rather than just reading numbers. We built a solid foundation for this, but adding more advanced interactive features would take it to the next level. Features like letting the engineers stack old test runs on top of each other, automatically highlight unexpected data spikes or have custom setups for different test types would make it significantly easier for the engineers to spot weird system behaviours.

Finally, once those advanced visualization features are added, the next step would be conducting formal and structured usability testing with the engineers. Getting the prototype for official testing rounds would ensure that the new visual tools correctly support their daily workflow.

8

Conclusion

This project addressed the need for specialized tools in industrial equipment testing by developing a software interface designed to help engineers explore massive amounts of live data. Specifically, this thesis aimed to answer the central research question: *What should be considered when designing interactive simulation systems for industrial equipment to support system verification through interactive information visualization?*

Using a Research through Design (RtD) approach inspired by the Double Diamond framework, this project explored the daily workflows and challenges of expert test engineers through semi-structured interviews and thematic analysis. An initial set of guidelines was created, and the Viper application was designed based on those guidelines. The guidelines were then refined and complemented, based on experience from the development phase and an evaluation using the initial guidelines. This resulted in a proposed set of guidelines that in addition to standard usability checklists that are too broad and generic for the specific demands of testing and verifying industrial process systems, should be used.

To effectively answer the research question, this thesis proposes that designers must look beyond traditional heuristics and combine foundational HCI principles with Information Visualization (InfoVis) theory. To meet these specific demands, this thesis proposes six core design guidelines:

- **Spatial Fidelity and Dynamic State Mapping:** Use visual models that reflect the actual hardware, saving users from having to constantly decode raw numbers.
- **Flexible Test Control:** Let engineers easily swap between running a script and taking manual control, without the system getting in their way.
- **Stable Views for Live Data:** Keep the screen locked exactly where the user is looking during an inspection, even while new data continues to stream in the background.
- **Clear Error Handling:** Clearly highlight different types of system messages and log them automatically, taking the burden of saving error data off the engineer.
- **Proactive Input Validation and Defensive Safety Nets:** Act as an active filter that blocks invalid data and conflicting commands before they execute,

rather than relying solely on the user's expertise to prevent critical errors.

- **Data Filtering:** Let the user freely select which data streams to display, and allow the user to save commonly used filter configurations.

These guidelines were translated into a functional web based software prototype with visual tank models, live data plotting, and manual control modes. Evaluation results confirmed that this interface effectively supports expert test engineers by presenting live hardware data in a highly interpretable format. The current system performs well in managing complex testing and ensuring stable data views without losing incoming information. However, future updates need stricter input validation to prevent conflicting commands from running at the same time. The system needs to provide clearer error messages so that engineers know exactly what went wrong and how to fix it. Additionally, future iteration should focus on improving accessibility with light theme and integrating more advanced visual analytics to help engineers easily compare historical test runs.

Ultimately, this project argues that to successfully support system verification, interactive simulation software must prioritize five core elements: accurate visual mapping, clear operational feedback, stable live data, uncompromising user control, and proactive system safety. However, at this time, the partner company has not yet decided what the next steps for the software will be.

Bibliography

- [1] P. Bishop, “Does software have to be ultra reliable in safety critical systems?” In *International Conference on Computer Safety, Reliability, and Security (SAFECOMP)*, ser. Lecture Notes in Computer Science, vol. 8153, Springer, 2013, pp. 118–129. DOI: 10.1007/978-3-642-40793-2_11.
- [2] J. Zhang, H. Ren, H. Ren, Y. Chai, Z. Liu, and X. Liang, “Comprehensive review of safety studies in process industrial systems: Concepts, progress, and main research topics,” *Processes*, vol. 11, no. 8, p. 2454, 2023. DOI: 10.3390/pr11082454.
- [3] Emerson Automation Solutions, *The Engineer’s Guide to Level Measurement*. 2021.
- [4] C. Ware, “Chapter one - foundations for an applied science of data visualization,” in *Information Visualization (Fourth Edition)*, ser. Interactive Technologies, C. Ware, Ed., Fourth Edition, Morgan Kaufmann, 2021, pp. 1–29, ISBN: 978-0-12-812875-6. DOI: <https://doi.org/10.1016/B978-0-12-812875-6.00001-3>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/B9780128128756000013>.
- [5] Aras Corporation. “Process industry.” Accessed: 2026-02-04. [Online]. Available: <https://aras.com/en/glossary/process-industry>.
- [6] Motilde. “Understanding the role of distributed control systems in industrial automation.” Published 2 October 2025; Accessed: 2026-02-04, Motilde. [Online]. Available: <https://motilde.com/en/understanding-role-distributed-control-systems-industrial-automation/>.
- [7] Emerson Electric Co. “Tank gauging systems and solutions,” Accessed: Jun. 16, 2026. [Online]. Available: <https://www.emerson.com/en/measurement-instrumentation/catalog/engineered-systems-and-solutions/tank-gauging#perPage=15&sortCriteria=relevance>.
- [8] Y. Zhao et al., “A visual analysis approach for understanding durability test data of automotive products,” *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 6, Dec. 2019, ISSN: 2157-6904. DOI: 10.1145/3345640. [Online]. Available: <https://doi.org/10.1145/3345640>.
- [9] J. Eirich et al., “Irvine: Using interactive clustering and labeling to analyze correlation patterns: A design study from the manufacturing of electrical engines,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 28, no. 1, pp. 11–21, Jan. 2022. DOI: 10.1109/TVCG.2021.3114797.

- [10] R. Splechtna et al., “Interactive visual analysis of structure-borne noise data,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 29, no. 1, pp. 778–787, 2023. DOI: 10.1109/TVCG.2022.3209478.
- [11] T. Funken and T. Meisen, “Visual analytics for industrial sensor data analysis,” Jan. 2021, pp. 584–593. DOI: 10.5220/0010399705840593.
- [12] O. Sindiy, K. Litomisky, S. Davidoff, and F. Dekens, “Introduction to information visualization (infovis) techniques for model-based systems engineering,” *Procedia Computer Science*, vol. 16, pp. 49–58, 2013. DOI: 10.1016/j.procs.2013.01.006.
- [13] J.-D. Fekete, “The infovis toolkit,” in *IEEE Symposium on Information Visualization*, IEEE, 2004, pp. 167–174.
- [14] Interaction Design Foundation, *Ux design*, <https://www.interaction-design.org/literature/topics/ux-design>, Accessed: 2026-02-03, 2026.
- [15] A. H. Allam, A. R. Che Hussin, and H. Mohamed Dahlan, “User experience: Challenges and opportunities,” *Journal of Information Systems Research and Innovation (JISRI)*, Accessed: 2026-02-03.
- [16] Interaction Design Foundation, *Interaction design (ixd)*, <https://www.interaction-design.org/literature/topics/interaction-design>, Accessed: 2026-02-03, 2026.
- [17] International Organization for Standardization (ISO), *Ergonomics of human-system interaction – part 11: Usability: Definitions and concepts*, International standard on usability in humansystem interaction, Geneva, Switzerland, 2018. DOI: 10.3403/iso9241-11:2018.
- [18] J. Nielsen, *Usability Engineering*. Morgan Kaufmann, 1993.
- [19] J. Preece, Y. Rogers, and H. Sharp, *Interaction Design: Beyond Human-Computer Interaction*, 4th ed. Wiley, 2015.
- [20] World Wide Web Consortium (W3C), *About the world wide web consortium (w3c)*, Accessed: 2026-03-23, 2024. [Online]. Available: <https://www.w3.org/about/>.
- [21] World Wide Web Consortium (W3C), *Web content accessibility guidelines (wcag) overview*, Accessed: 2026-03-23, 2018. [Online]. Available: <https://www.w3.org/WAI/standards-guidelines/wcag/>.
- [22] World Wide Web Consortium (W3C), *About understanding wcag documents*, Accessed: 2026-03-23, 2018. [Online]. Available: <https://www.w3.org/WAI/WCAG21/Understanding/>.
- [23] S. K. Card, J. Mackinlay, and B. Shneiderman, *Readings in information visualization: using vision to think*. Morgan Kaufmann, 1999.
- [24] W. S. Cleveland and R. McGill, “Graphical perception: Theory, experimentation, and application to the development of graphical methods,” *Journal of the American statistical association*, vol. 79, no. 387, pp. 531–554, 1984.
- [25] M. Khan and Khan, “Data and information visualization methods, and interactive mechanisms: A survey,” *International Journal of Computer Applications*, vol. 34, pp. 1–14, Dec. 2011.
- [26] J. Wang, K. Kucher, R. Pates, and A. Kerren, “Euroenergyvis: Interactive visualization of power plant data for european countries,” in *Proceedings of the 18th International Symposium on Visual Information Communication and*

- Interaction*, ser. VINCI '25, Association for Computing Machinery, 2025, ISBN: 9798400718458. DOI: 10.1145/3769534.3769541. [Online]. Available: <https://doi.org/10.1145/3769534.3769541>.
- [27] M. Rester and M. Pohl, “Methods for the evaluation of an interactive infovis tool supporting exploratory reasoning processes,” in *BELIV 2006 Venice, Italy*, ACM, 2006.
- [28] The Editors of Encyclopædia Britannica. “Scientific visualization.” most recently revised and updated by Erik Gregersen, Encyclopædia Britannica. [Online]. Available: <https://www.britannica.com/technology/scientific-visualization>.
- [29] A. C. Telea, *Data Visualization: Principles and Practice*, 2nd. CRC Press, 2014.
- [30] H. W. Rittel and M. M. Webber, “Dilemmas in a general theory of planning,” *Policy Sciences*, vol. 4, no. 2, pp. 155–169, 1973.
- [31] A. Cooper, R. Reimann, D. Cronin, and C. Noessel, *About Face: The Essentials of Interaction Design*, 4th ed. Indianapolis, IN: John Wiley & Sons, 2014, ISBN: 978-1-118-76657-6.
- [32] J. Nielsen and T. K. Landauer, “A mathematical model of the finding of usability problems,” in *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, ACM, 1993, pp. 206–213. DOI: 10.1145/169059.169166.
- [33] J. Nielsen, “Why you only need to test with 5 users,” *Nielsen Norman Group*, 2000. [Online]. Available: <https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/>.
- [34] P. J. Stappers and E. Giaccardi, “Research through design,” in *The Encyclopedia of Human-Computer Interaction, 2nd Ed.* Interaction Design Foundation, 2014. [Online]. Available: <https://ixdf.org/literature/book/the-encyclopedia-of-human-computer-interaction-2nd-ed/research-through-design>.
- [35] J. Zimmerman, J. Forlizzi, and S. Evenson, “Research through design as a method for interaction design research in HCI,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '07, San Jose, California, USA: Association for Computing Machinery, 2007, pp. 493–502, ISBN: 9781595935939. DOI: 10.1145/1240624.1240704. [Online]. Available: <https://doi.org/10.1145/1240624.1240704>.
- [36] Design Council, *Framework for innovation*, <https://www.designcouncil.org.uk/our-resources/framework-for-innovation/>, Accessed: 2026-02-03, 2026.
- [37] J. A. Marin-Garcia, J. J. Garcia-Sabater, J. P. Garcia-Sabater, and J. Maheut, “Protocol: Triple diamond method for problem solving and design thinking. rubric validation,” *WPOM - Working Papers on Operations Management*, vol. 11, no. 2, pp. 49–68, Dec. 2020. DOI: 10.4995/wpom.v11i2.14776. [Online]. Available: <https://polipapers.upv.es/index.php/WPOM/article/view/14776>.

- [38] D. Council. "The double diamond." Accessed: 2026-02-04. [Online]. Available: <https://www.designcouncil.org.uk/our-resources/the-double-diamond/>.
- [39] D. Gustafsson, "Designing loudspeakers: From prototyping to final product," Master's thesis, Aalto University, School of Arts, Design and Architecture, 2019. Accessed: Feb. 3, 2026. [Online]. Available: <https://aaltodoc.aalto.fi/handle/123456789/39285>.
- [40] J. Rattray and M. C. Jones, "Essential elements of questionnaire design and development," *Journal of Clinical Nursing*, vol. 16, no. 2, pp. 234–243, 2007. DOI: <https://doi.org/10.1111/j.1365-2702.2006.01573.x>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1365-2702.2006.01573.x>. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1365-2702.2006.01573.x>.
- [41] C. Wilson, "Chapter 1 - structured interviews," in *Interview Techniques for UX Practitioners*, C. Wilson, Ed., Boston: Morgan Kaufmann, 2014, pp. 1–21, ISBN: 978-0-12-410393-1. DOI: <https://doi.org/10.1016/B978-0-12-410393-1.00001-6>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/B9780124103931000016>.
- [42] C. Wilson, "Chapter 2 - semi-structured interviews," in *Interview Techniques for UX Practitioners*, C. Wilson, Ed., Boston: Morgan Kaufmann, 2014, pp. 23–41, ISBN: 978-0-12-410393-1. DOI: <https://doi.org/10.1016/B978-0-12-410393-1.00002-8>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/B9780124103931000028>.
- [43] C. Wilson, "Chapter 3 - unstructured interviews," in *Interview Techniques for UX Practitioners*, C. Wilson, Ed., Boston: Morgan Kaufmann, 2014, pp. 43–62, ISBN: 978-0-12-410393-1. DOI: <https://doi.org/10.1016/B978-0-12-410393-1.00003-X>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/B978012410393100003X>.
- [44] C. Wilson, "Chapter 5 - focus groups," in *Interview Techniques for UX Practitioners*, C. Wilson, Ed., Boston: Morgan Kaufmann, 2014, pp. 83–111, ISBN: 978-0-12-410393-1. DOI: <https://doi.org/10.1016/B978-0-12-410393-1.00005-3>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/B9780124103931000053>.
- [45] T. George. "Structured interview: Definition, guide & examples." Revised on 18 July 2024. [Online]. Available: <https://www.scribbr.co.uk/research-methods/structured-interviews/>.
- [46] H. Taherdoost, "Sampling methods in research methodology; how to choose a sampling technique for research," *How to choose a sampling technique for research (April 10, 2016)*, 2016.
- [47] S. K. Ahmed et al., "Using thematic analysis in qualitative research," *Journal of Medicine, Surgery, and Public Health*, vol. 6, p. 100198, 2025, ISSN: 2949-916X. DOI: <https://doi.org/10.1016/j.glmedi.2025.100198>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2949916X25000222>.
- [48] Google, *Crazy 8's: Phase 3 sketch*, <https://designsprintkit.withgoogle.com/methodology/phase3-sketch/crazy-8s>, Accessed: 2026-02-25, n.d.

-
- [49] R. Krause, *Storyboards help visualize ux ideas*, Accessed: 2026-02-25, Nielsen Norman Group, Jul. 2018. [Online]. Available: <https://www.nngroup.com/articles/storyboards-visualize-ideas/>.
 - [50] E. de Bono, *Six Thinking Hats*. Boston: Little, Brown and Company, 1985.
 - [51] Design Council, *Design methods for developing services*, Original publication by the Design Council, 2015. [Online]. Available: <https://idoc.pub/documents/design-methods-for-developing-servicespdf-qn85m7prd2n1>.
 - [52] B. Buxton, *Sketching User Experiences: Getting the Design Right and the Right Design*. Morgan Kaufmann, 2007.
 - [53] J. Rubin and D. Chisnell, *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests*, 2nd. Wiley, 2008.
 - [54] K. A. Ericsson and H. A. Simon, *Protocol Analysis: Verbal Reports as Data*, Rev. MIT Press, 1993.
 - [55] M. van den Haak, M. de Jong, and P. J. Schellens, “Retrospective vs. concurrent think-aloud protocols: Testing the usability of an online library catalogue,” *Behaviour & Information Technology*, vol. 22, no. 5, pp. 339–351, 2003.
 - [56] J. Nielsen, *Usability Engineering*. San Francisco, CA: Morgan Kaufmann, 1994.
 - [57] B. Shneiderman, *The eight golden rules of interface design*, <https://www.cs.umd.edu/~ben/goldenrules.html>, Accessed: 2026-05-06, University of Maryland, Department of Computer Science, 2016.
 - [58] K. Schwaber and J. Sutherland, *The scrum guide*, Accessed: 2026-02-10, 2020. [Online]. Available: <https://scrumguides.org/scrum-guide.html>.
 - [59] R. Pedersen and M. May, “Visualization in process control,” *Human Factors and Ergonomics in Manufacturing & Service Industries*, vol. 9, no. 3, pp. 267–276, 1999. DOI: [https://doi.org/10.1002/\(SICI\)1520-6564\(199922\)9:3<267::AID-HFM4>3.0.CO;2-J](https://doi.org/10.1002/(SICI)1520-6564(199922)9:3<267::AID-HFM4>3.0.CO;2-J). eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/%28SICI%291520-6564%28199922%299%3A3%3C267%3A%3AAID-HFM4%3E3.0.CO%3B2-J>. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291520-6564%28199922%299%3A3%3C267%3A%3AAID-HFM4%3E3.0.CO%3B2-J>.

A

Appendix 1

A.1 Interview Guiding Questions

1. Can you walk me through the test environments you currently use in your daily work?
2. What works well in your current test environments, and what aspects cause friction or could be improved?
3. What are the biggest challenges you face when working in your test environment?
4. What factors most often slow down your test execution or verification workflows?
5. How frequently do you run tests in the test environment?
6. How do you imagine the IPE workflow or want it to be like?

A.2 How Might We Questions

- **Error Handling & Detection**

- *How might we help users notice and understand errors earlier?*
 - * Show warnings or feedback directly in the interface while users work.
 - * Highlight potential issues before they become real errors.
 - * Make error messages clearer and easier to understand.
 - * Help users see exactly where an error occurred.
- *How might we make it easier to understand what caused an error?*
 - * Provide short explanations of what triggered the issue.
 - * Show how different steps or inputs are connected.
 - * Allow users to review what happened right before the error.

- **Flexibility**

- *How might we show advanced options without overwhelming users?*
 - * Reveal advanced settings only when needed.
 - * Allow switching between a simple and an advanced view.
 - * Adjust the interface based on user behaviour.
- *How might we give experienced users more control?*
 - * Allow users to create custom workflows.
 - * Provide a scripting or automation option.
 - * Add step-by-step controls for more detailed operation.
- **Intuitiveness**
 - *How might we make the system easier for new users?*
 - * Offer a short guided introduction with examples.
 - * Use helpful default settings to reduce early decisions.
 - * Provide small hints when users seem unsure.
 - *How might we guide users through complex workflows?*
 - * Break larger tasks into smaller, guided steps.
 - * Use checklists to show progress and next steps.
 - * Add contextual tips when needed.
- **Automation & Convenience**
 - *How might we support automation for repetitive tasks?*
 - * Let users queue multiple tasks to run automatically.
 - * Provide ready-made templates for common workflows.
 - * Allow tasks to run in the background with notifications.
 - *How might we allow manual control when needed?*
 - * Add a panel for adjusting settings during a process.
 - * Include pause, resume, and restart controls.
 - * Support a preview mode before running actions.
- **Analysis Tools**
 - *How might we help users examine their data more effectively?*
 - * Allow filtering, sorting, and rearranging data.
 - * Suggest useful ways to explore the data.
 - * Provide comparison views.

- *How might we make visualizations more interactive?*
 - * Let users click on elements to see more details.
 - * Allow marking or annotating graphs.
 - * Update visualizations instantly when settings change.