A WEB-BASED WIZARD-OF-OZ PLATFORM FOR COLLABORATIVE AND REPRODUCIBLE HUMAN-ROBOT INTERACTION RESEARCH

by

Sean O'Connor

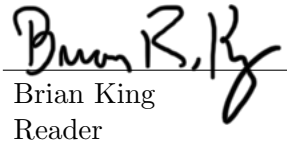
A Thesis

Presented to the Faculty of
Bucknell University
in Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science with Honors in Computer Science and Engineering

April 21, 2026

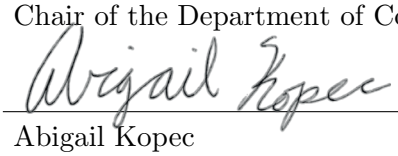
Approved:

L. Felipe Perrone
Thesis Advisor



Brian King
Reader

Alan Marchiori
Chair of the Department of Computer Science



Abigail Kopec
Honors Council Representative

Acknowledgments

I owe a great deal to my advisor, Felipe Perrone. From my first year at Bucknell, he gave me the resources and the space to learn on my own terms, and has supported me at every turn — inside the classroom and far beyond it. The many credits I spent in his courses; the countless hours spent meeting, reading, and rereading drafts together; the two papers we wrote and the trips we took to present them; all of it built the skills and the judgment that this thesis required, and much more besides. I could not have done this without him.

I also thank Professor Brian King, whose encouragement of curiosity and discovery has stayed with me, and whose courses have given me the technical foundation to take on projects with confidence and lead them well. I am glad to have him on my committee.

I thank Professor Kopec for her service as the Honors Council representative on my thesis committee, and the six Bucknell faculty members who volunteered their time to participate in the pilot study that made this evaluation possible.

My parents have supported me throughout my time at Bucknell and in everything that came before it. This is for them, and for my grandfather, whose journey here made mine possible.

Contents

List of Tables

List of Figures

Abstract

The Wizard-of-Oz (WoZ) technique is widely used in Human-Robot Interaction (HRI) research, but two persistent problems limit its effectiveness: existing tools impose technical barriers that exclude non-engineering domain experts (the Accessibility Problem), and the fragmented landscape of robot-specific implementations makes interaction scripts difficult to port across platforms (the Reproducibility Problem — concerning execution consistency and portability, not third-party replication). Through a literature review, I identified three design principles to address both: a hierarchical specification model, an event-driven execution model, and a plugin architecture that decouples experiment logic from robot-specific implementations. I realized these principles in HRIStudio, an open-source, web-based platform providing a visual experiment designer, a guided wizard execution interface, automated timestamped logging with deviation tracking, and role-based access control.

I evaluated HRIStudio in a pilot between-subjects study (N=6) against Choregraphe, the standard programming tool for the NAO robot. HRIStudio wizards achieved higher design fidelity, execution reliability, and perceived usability across all six sessions; the only unprompted specification deviation in the dataset occurred in the Choregraphe condition. While the pilot scale precludes inferential claims, the directional evidence across all measures supports the position that a tool built to realize the identified design principles can have significant impact on accessibility and reproducibility in WoZ-based HRI research.

Chapter 1

Introduction

Human-Robot Interaction (HRI) is an essential field of study for understanding how robots should communicate, collaborate, and coexist with people. As researchers work to develop social robots capable of natural interaction, they face a fundamental challenge: how to prototype and evaluate interaction designs before the underlying autonomous systems are fully developed. This chapter introduces the technical and methodological barriers that currently limit Wizard-of-Oz (WoZ) based HRI research, describes a generalized approach to address these challenges, and establishes the research objectives and thesis statement for this work.

1.1 Motivation

To build the social robots of tomorrow, researchers must study how people respond to robot behavior today. That requires interactions that feel real even when autonomy

is incomplete. The process of designing and optimizing interactions between human and robot is essential to HRI, a discipline dedicated to ensuring these technologies are safe, effective, and accepted by the public [1]. However, current practices for prototyping these interactions are often hindered by complex technical requirements and inconsistent methodologies.

Social robotics, a subfield of HRI, focuses on robots designed for social interaction with humans, and it poses unique challenges for autonomy. In a typical social robotics interaction, a robot operates autonomously based on pre-programmed behaviors. Because human reactions to robot behaviors are not always predictable, pre-programmed autonomy often fails to respond appropriately to subtle social cues, causing the interaction to degrade.

To overcome this limitation, researchers use the WoZ technique. The name references L. Frank Baum's story [2], in which the "great and powerful" Oz is revealed to be an ordinary person operating machinery behind a curtain, creating an illusion of magic. In WoZ experiments, the wizard similarly creates an illusion of robot intelligence from behind the scenes. Consider a scenario where a researcher wants to test whether a robot tutor can effectively encourage student subjects during a learning task. Rather than building a complete autonomous system with speech recognition, natural language understanding, and emotion detection, the researcher may use a WoZ setup: a human operator (the "wizard") sits in a separate room, observing the interaction through cameras and microphones. When the subject appears frustrated, the wizard makes the robot say an encouraging phrase and perform a supportive gesture. To the subject, the robot appears to be acting autonomously, responding naturally to the subject's emotional state. This methodology allows researchers to

rapidly prototype and test interaction designs, gathering valuable data about human responses before investing in the development of complex autonomous capabilities.

Despite its versatility, WoZ research faces two critical challenges. The first is *The Accessibility Problem*: many non-programmers, such as experts in psychology or sociology, may find it challenging to conduct their own studies without engineering support. The second is *The Reproducibility Problem*: the hardware landscape is highly fragmented, and researchers frequently build custom control interfaces for specific robots and experiments. Because these tools are tightly coupled to particular hardware, running the same social interaction script on a different robot platform typically requires rebuilding the implementation from scratch. These tools are rarely shared, making it difficult for a researcher to reproduce the same study across different robot platforms or for other labs to replicate results.

1.2 Proposed Approach

To address the accessibility and reproducibility problems in WoZ-based HRI research, I propose a web-based software framework that integrates three key capabilities. First, the framework must provide an intuitive interface for experiment design that does not require programming expertise, enabling domain experts from psychology, sociology, or other fields to create interaction protocols independently. Second, it must enforce methodological rigor during experiment execution by guiding the wizard through standardized procedures and preventing deviations from the experimental script that could compromise validity. Third, it must be platform-agnostic, meaning the same experiment design can be reused across different robot hardware as technology evolves.

This approach represents a shift from the current paradigm of custom, robot-specific tools toward a unified platform that can serve as shared infrastructure for WoZ-based HRI research. By treating experiment design, execution, and analysis as distinct but integrated phases of a study, such a framework can systematically address both technical barriers and sources of variability that currently limit research quality and reproducibility.

The contributions of this thesis are the design principles of this approach, namely: a hierarchical specification model, an event-driven execution model, and a plugin architecture that decouples experiment logic from robot-specific implementations. Together they form a coherent architecture for WoZ infrastructure that any implementation could adopt. To evaluate the impact of these design principles, I developed a reference implementation called HRIStudio: an open-source, web-based platform built on this architecture and used as the instrument for empirical validation.

1.3 Research Objectives

This thesis builds upon foundational work presented in two prior peer-reviewed publications. Prof. Perrone and I first introduced the conceptual framework for HRIStudio at the 2024 IEEE International Conference on Robot and Human Interactive Communication (RO-MAN) [3], establishing the vision for a collaborative, web-based platform. Subsequently, we published the detailed system architecture and a first prototype at RO-MAN 2025 [4], validating the technical feasibility of web-based robot control. Those publications established our vision and a first prototype. This thesis extends and formalizes our contributions: a set of design principles for WoZ infras-

tructure that simultaneously address the *Accessibility* and *Reproducibility* Problems, a reference implementation that realizes those principles, and pilot empirical evidence that they produce measurably different outcomes in practice.

The central question this thesis addresses is: *can the right software architecture make Wizard-of-Oz experiments more accessible to non-programmers and more reproducible across participants?* To answer it, I propose a hierarchical, event-driven specification model that separates protocol design from trial execution, enforces action sequences, and logs deviations automatically; implement it as HRIStudio; and evaluate it in a pilot study comparing design fidelity and execution reliability against a representative baseline tool. The goal is not to prove a statistical effect at scale, but to establish directional evidence that the architecture changes what researchers can do and guides them to be consistent in the pursuit of their experimental goals.

1.4 Chapter Summary

This chapter has established the context and objectives for this thesis. I identified two critical challenges facing WoZ-based HRI research. The first is the *Accessibility Problem*: high technical barriers limit participation by non-programmers. The second is the *Reproducibility Problem*: fragmented tooling makes results difficult to replicate across labs. I proposed a web-based framework approach that addresses these challenges through intuitive design interfaces, enforced experimental protocols, and platform-agnostic architecture. Finally, I posed the central research question and described how this thesis addresses it through formal design, a reference implementation, and a pilot validation study.

Chapter 2

Background and Related Work

This chapter provides the necessary context for understanding the challenges addressed by this thesis. I survey the landscape of existing WoZ platforms, analyze their capabilities and limitations, and establish requirements that a modern infrastructure should satisfy. Finally, I position this thesis within the context of prior work on this topic.

As established in Chapter 1, the WoZ technique enables researchers to prototype and test robot interaction designs before autonomous capabilities are developed. This thesis is situated within a specific subset of HRI activity: social robotics, a subfield concerned with robots designed for direct social interaction with humans, and more narrowly within that, WoZ experiments used to prototype and evaluate social robot behaviors. To understand how the proposed framework advances this research paradigm, I review the existing landscape of WoZ platforms, identify their limitations relative to disciplinary needs, and establish requirements for a more comprehensive

approach. HRI is fundamentally a multidisciplinary field which brings together engineers, psychologists, designers, and domain experts from various application areas [1]. Yet two challenges have historically limited participation from non-technical researchers in WoZ-based HRI studies. First, high technical barriers prevent many domain experts from conducting independent studies. Second, each research group builds custom software for specific robots, creating tool fragmentation across the field.

2.1 Existing WoZ Platforms and Tools

Over the last two decades, multiple frameworks to support and automate the WoZ paradigm have been reported in the literature. These frameworks can be broadly categorized based on their primary design emphases, generality, and the methodological practices they encourage. Foundational work by Steinfeld et al. [5] articulated the methodological importance of WoZ simulation, distinguishing between the human simulating the robot (Wizard of Oz) and the robot simulating the human. In the latter case (Oz of Wizard), the robot acts as if controlled by a person when it is actually autonomous. This distinction has influenced how subsequent tools approach the design and execution of WoZ experiments.

Early platform-agnostic tools focused on providing robust, flexible interfaces for technically sophisticated users. These systems were designed to work with multiple robot types rather than a single hardware platform. Polonius [6], built on the Robot Operating System (ROS) [7], exemplifies this generation. It provides a graphical interface for defining finite state machine scripts that control robot behaviors, with integrated logging capabilities to streamline post-experiment analysis. The system

was explicitly designed to enable robotics engineers to create experiments that their non-technical collaborators could then execute. However, the initial setup and configuration still required substantial programming expertise. Similarly, OpenWoZ [8] introduced a cloud-based, runtime-configurable architecture using web protocols. Its design allows multiple operators or observers to connect simultaneously, and its plugin system enables researchers to extend functionality such as adding new robot behaviors or sensor integrations. Most importantly, OpenWoZ allows runtime modification of robot behaviors, enabling wizards to deviate from scripts when unexpected situations arise. While architecturally sophisticated and highly flexible, OpenWoZ requires programming knowledge to create custom behaviors and configure experiments, creating the *Accessibility Problem* for non-technical researchers.

A second wave of tools shifted focus toward usability, often achieving accessibility by coupling tightly with specific hardware platforms. WoZ4U [9] was explicitly designed as an "easy-to-use" tool for conducting experiments with Aldebaran's Pepper robot. It provides an intuitive graphical interface that allows non-programmers to design interaction flows, and it successfully lowers the technical barrier. However, this usability comes at the cost of generalizability. WoZ4U does not work for other robot platforms. Manufacturer-provided software for various robots follow a similar pattern.

Choregraphe [10], developed by Aldebaran Robotics for the NAO and Pepper robots, offers a visual programming environment based on connected behavior boxes. Researchers can create complex interaction flows using drag-and-drop blocks without writing code in traditional programming languages. However, when new robot platforms emerge or when hardware becomes obsolete, tools like Choregraphe and

WoZ4U lose their utility. Pettersson and Wik, in their review of WoZ tools [11], note that platform-specific systems often fall out of use as technology evolves, forcing researchers to constantly rebuild their experimental infrastructure.

Recent years have seen renewed interest in comprehensive WoZ frameworks. Gibert et al. [12] developed the Super Wizard of Oz (SWoOZ) platform. This system integrates facial tracking, gesture recognition, and real-time control capabilities to enable naturalistic human-robot interaction studies. Virtual and augmented reality have also emerged as complementary approaches to WoZ. Helgert et al. [13] demonstrated how VR-based WoZ environments can simplify experimental setup while providing researchers with precise control over environmental conditions and high-fidelity data collection.

This expanding landscape reveals a persistent fundamental gap in the design space of WoZ tools. Flexible, general-purpose platforms like Polonius and OpenWoZ offer powerful capabilities but present high technical barriers. Accessible, user-friendly tools like WoZ4U and Choregraphe lower those barriers but sacrifice cross-platform compatibility and longevity. Newer approaches such as VR-based frameworks attempt to bridge this gap, yet no existing tool successfully combines accessibility, flexibility, deployment portability, and built-in methodological rigor.

The missing quadrant in Figure 2.1 matters because methodological rigor requires systematic features that guide experimenters toward best practices: consistently following experimental protocols, maintaining comprehensive logging, and producing reproducible experimental designs. Few platforms directly address the methodological concerns raised by systematic reviews of WoZ research. Riek's influential analysis [14] of 54 HRI studies uncovered widespread inconsistencies in how wizard behav-

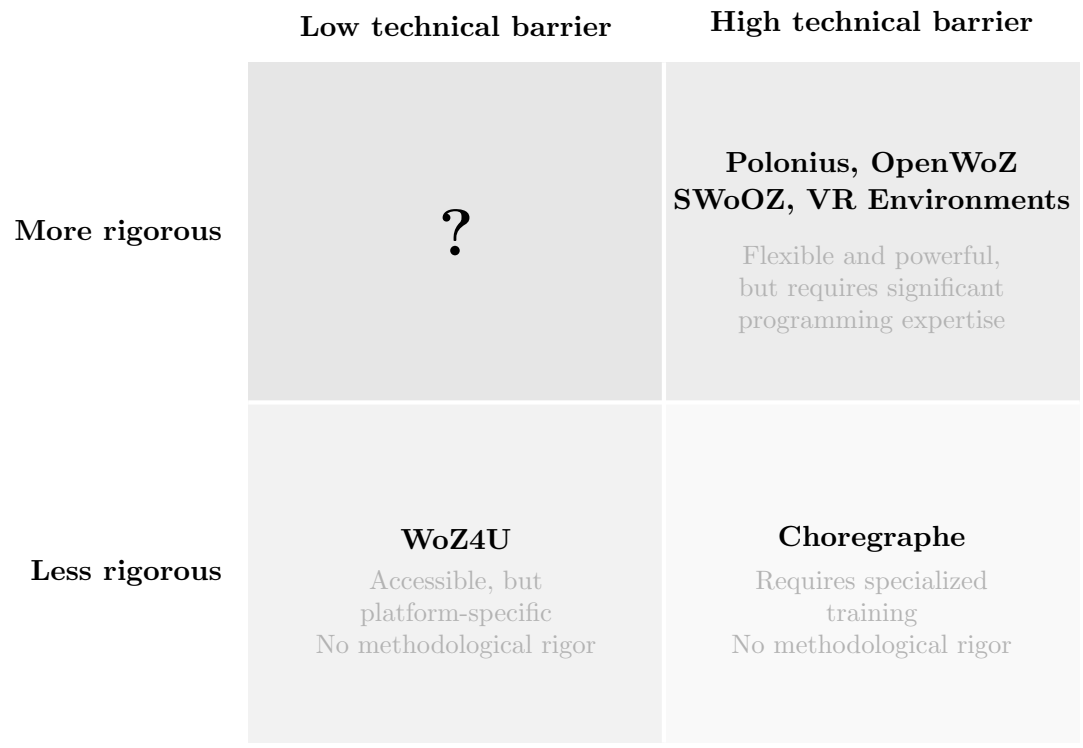


Figure 2.1: WoZ tool design space by technical barrier and methodological rigor.

iors were controlled and reported. Very few studies documented standardized wizard training procedures or measured wizard error rates, raising questions about internal validity—that is, whether observed outcomes can be attributed to the intended experimental manipulation rather than to uncontrolled variation in wizard behavior. The tools themselves often exacerbate this problem: poorly designed interfaces increase cognitive load on wizards, leading to timing errors and behavioral inconsistencies that can confound experimental results. Recent work by Strazdas et al. [15] further demonstrates the importance of careful interface design in WoZ systems, showing that intuitive wizard interfaces directly improve both the quality of robot behavior and the reliability of collected data.

2.2 Requirements for Modern WoZ Infrastructure

This thesis is the latest step in a multi-year effort to build infrastructure that addresses the challenges identified in the WoZ platform landscape. Based on the analysis of existing platforms and identified methodological gaps, I derived requirements for a modern WoZ research infrastructure. Through our preliminary work [3], we identified six critical capabilities that a comprehensive platform should provide:

R1: Integrated workflow. All phases of the experimental workflow (design, execution, and analysis) should be integrated within a single unified environment, so that researchers do not need to move between separate tools to design, run, and analyze their experiments.

R2: Low technical barrier. Creating social interaction protocols between robot and human should require minimal to no programming expertise, enabling domain experts from psychology, education, or other fields to work independently [1].

R3: Real-time control. The system must support fine-grained, responsive real-time control during live experiment sessions across a variety of robotic platforms. Consistent real-time control across platforms also directly supports reproducibility: the same script should execute with equivalent responsiveness regardless of which robot is used.

R4: Automated logging. All actions, timings, and sensor data should be automatically logged with synchronized timestamps to facilitate analysis.

R5: Platform agnosticism. The architecture should decouple experimental logic

from robot-specific actions and behaviors. This allows experiments designed for one robot platform to be adapted to others, ensuring the WoZ platform remains viable as hardware evolves. This requirement also directly addresses the Reproducibility Problem: a platform-agnostic design makes it possible to run the same interaction script on different robots with minimal change to the implementing program.

R6: Collaborative support. Multiple team members should be able to contribute to experiment design and review of execution data, supporting truly interdisciplinary research.

To the best of my knowledge, no existing platform satisfies all six requirements. Most critically, the trade-off between accessibility and reproducibility remains unresolved. Few tools embed methodological best practices directly into their design to guide experimenters toward sound methodology by default.

This work builds on two prior peer-reviewed publications. We first introduced the concept for HRIStudio as a Late-Breaking Report at the 2024 IEEE International Conference on Robot and Human Interactive Communication (RO-MAN) [3]. In that position paper, we identified the lack of accessible tooling as a primary barrier to entry in HRI and proposed the high-level vision of a web-based, collaborative platform. We established the core requirements listed above and argued for a web-based approach to achieve them.

Following the initial proposal, we published the detailed system architecture and preliminary prototype as a full paper at RO-MAN 2025 [4]. That publication validated the technical feasibility of our approach, detailing the communication protocols, data

models, and plugin architecture necessary to support real-time robot control using standard web technologies while maintaining platform independence.

While those prior publications established the conceptual framework and technical architecture, this thesis formalizes those design principles, realizes them in a complete implementation, and evaluates whether they produce measurably different outcomes in a pilot validation study. The pilot study compares design fidelity and execution reliability between HRIStudio and a representative baseline tool, showing whether these principles translate into better outcomes for real researchers.

2.3 Chapter Summary

This chapter has established the technical and methodological context for this thesis. Existing WoZ platforms fall into two categories: general-purpose tools like Polonius and OpenWoZ that offer flexibility but high technical barriers, and platform-specific systems like WoZ4U and Choregraphe that prioritize usability at the cost of cross-platform generality. Recent approaches such as VR-based frameworks attempt to bridge this gap, yet to the best of my knowledge, no existing tool successfully combines accessibility, flexibility, and embedded methodological rigor. Based on this landscape analysis, I identified six critical requirements for modern WoZ infrastructure (R1–R6): integrated workflows, low technical barriers, real-time control across platforms, automated logging, platform-agnostic design, and collaborative support. These requirements are the standard against which the proposed design is evaluated in Chapter 6. The next chapter examines the broader reproducibility challenges that justify why these requirements are essential.

Chapter 3

Reproducibility Challenges

Having established the landscape of existing WoZ platforms and their limitations, I now examine the factors that make WoZ experiments difficult to reproduce consistently and how software infrastructure can address them. This chapter analyzes the sources of variability identified in the WoZ literature and examines how current practices in infrastructure and reporting contribute to *the Reproducibility Problem*. Understanding these challenges is essential for designing a system that supports reproducible, rigorous experimentation.

3.1 Sources of Variability

The Reproducibility Problem, as introduced in Chapter 1, encompasses two related challenges. The first concerns *execution consistency*: whether a wizard reliably follows the same experimental script across multiple trials with different participants,

producing comparable robot behavior in each. The second concerns *cross-platform reproducibility*: whether the same experiment can be transferred to a different robot platform with minimal change to the implementing program. Both stem from gaps in current WoZ infrastructure and are examined in this chapter. It is important to note that the term reproducibility may also refer to *allowing independent replications of published studies*; this is not what this thesis evaluates. Execution consistency, as defined here, corresponds to what the measurement literature sometimes calls *repeatability*: the degree to which the same procedure produces consistent results when repeated across multiple trials of the same study.

In WoZ-based HRI studies, multiple sources of variability can compromise execution consistency. The wizard is simultaneously the strength and weakness of the WoZ paradigm. While human control enables sophisticated, adaptive interactions, it also introduces inconsistency. Consider a wizard conducting multiple trials of the same experiment with different participants. Even with a detailed script, the wizard may vary in timing, with the delay between a participant's action and the robot's response fluctuating based on the wizard's attention, fatigue, or interpretation of when to act. When a script allows for choices, different wizards may make different selections, or the same wizard may act differently across trials. Furthermore, a wizard may accidentally skip steps, trigger actions in the wrong order, or misinterpret experimental protocols.

Riek's systematic review [14] found that very few published studies reported measuring wizard error rates or providing standardized wizard training. Without such measures, it becomes impossible to determine whether experimental results reflect the intended interaction design or inadvertent variations in wizard behavior.

Beyond wizard behavior, the custom nature of many WoZ control systems introduces technical variability. When each research group builds custom software for each study, several problems arise. Custom interfaces may have undocumented capabilities, hidden features, default behaviors, or timing characteristics researchers never formally describe. Software tightly coupled to specific robot models or operating system versions may become unusable when hardware or software is upgraded or replaced. Each system logs data differently, with different file formats, different levels of granularity, and different choices about what to record. This fragmentation undermines both execution consistency and reproducibility. Rebuilding custom infrastructure for each study makes it nearly impossible to guarantee that wizard behavior is controlled the same way across trials. More broadly, reproducing the same experiment on a different robot platform typically requires reverse-engineering or rebuilding the original software from scratch.

Even when researchers intend for their work to be reproducible, practical constraints on publication length lead to incomplete documentation. Papers often omit exact timing parameters. Authors leave decision rules for wizard actions unspecified and fail to report details of the wizard interface. Specifications of data collection, including which sensor streams were recorded and at what sampling rate, frequently go missing. Without this information, other researchers cannot faithfully recreate the experimental conditions, limiting both direct replication and conceptual extensions of prior work.

3.2 Infrastructure Requirements for Enhanced Reproducibility

Based on this analysis, I identify specific ways that software infrastructure can mitigate reproducibility challenges:

1. **Guided wizard execution.** Rather than merely providing tools for wizard control, an ideal WoZ platform should actively guide wizards through scripted procedures. This means presenting actions in a prescribed sequence to prevent out-of-order execution, highlighting the current step in the protocol, recording any deviations from the script as explicit events in the data log, and supporting repeatable decision logic through clearly defined conditional branches. By constraining wizard behavior within the bounds of the experimental design, the system reduces unintended variability across trials and participants.

2. **Comprehensive automatic logging.** Manual data collection is error-prone and often incomplete. The platform should automatically record every action triggered by the wizard with precise timestamps, all robot sensor data and state changes, and timing information indicating when actions were requested, when they began executing, and when they completed. The full experimental protocol should be embedded in each log file so that researchers can recover the exact script used for any session. Note that recording precise timestamps does not imply that trials must have identical timing, since human-robot interactions naturally vary in duration; rather, the system captures what actually occurred for later analysis.

3. **Self-documenting protocol specifications.** The protocol specification itself

should serve as documentation. When interaction protocols are defined using structured formats such as visual flowcharts or declarative scripts rather than imperative code, they become simultaneously executable and human-readable. Researchers can then share complete, unambiguous descriptions of their experimental procedures alongside their results.

4. **Platform-independent abstractions.** To maximize the lifespan and transferability of experimental designs, the platform must separate the high-level control logic, the sequence of wizard and robot actions, from the low-level details of how specific robots execute those behaviors. This abstraction allows experiments designed for one robot to be more easily adapted to another, extending the reproducibility of interaction designs even when the original hardware becomes obsolete.

3.3 Connecting Reproducibility Challenges to Infrastructure Requirements

The reproducibility challenges identified above directly motivate the infrastructure requirements (R1–R6) established in Chapter 2. Inconsistent wizard behavior creates the need for real-time control mechanisms (R3) that guide wizards step by step, and for automatic logging (R4) that captures any deviations that occur. Timing errors further motivate responsive, fine-grained real-time control (R3): a wizard working with a sluggish interface introduces latency that disrupts the interaction and confounds timing analysis. Technical fragmentation forces each lab to rebuild infrastructure as hardware changes, violating platform agnosticism (R5). Incomplete documentation

reflects the need for self-documenting, code-free protocol specifications (R2) that are simultaneously executable and shareable, integrated into a single workflow (R1) so that the specification and the execution environment are never separated. Finally, the isolation of individual research groups motivates collaborative support (R6): allowing multiple team members to observe and review trials enables the shared scrutiny that reproducibility requires. As Chapter 2 demonstrated, no existing platform simultaneously satisfies all six requirements. Addressing this gap requires rethinking how WoZ infrastructure is designed, prioritizing reproducibility and methodological rigor as first-class design goals rather than afterthoughts.

3.4 Chapter Summary

This chapter has analyzed the reproducibility challenges inherent in WoZ-based HRI research, identifying three primary sources of variability: inconsistent wizard behavior, fragmented technical infrastructure, and incomplete documentation. Rather than treating these challenges as inherent to the WoZ paradigm, I showed how each stems from gaps in current infrastructure. Software design can systematically mitigate them through enforced experimental protocols, comprehensive automatic logging, self-documenting experiment designs, and platform-independent abstractions. These design goals directly address the six infrastructure requirements identified in Chapter 2. The following chapters describe the design, implementation, and pilot validation of a system that prioritizes reproducibility as a foundational design principle from inception.

Chapter 4

Architectural Design

Chapter 2 established six requirements for modern WoZ infrastructure, labeled R1 through R6, and Chapter 3 showed the reproducibility problems that motivate them. This chapter presents the architectural contribution of this thesis: a hierarchical specification model, an event-driven execution model, a modular interface architecture, and an integrated data flow that together address all six requirements. These are design principles, not implementation details; they apply to any system built with the same goals.

4.1 Hierarchical Organization of Experiments

WoZ studies involve multiple experiments, shared protocol phases, and platform-specific behaviors that span the full research lifecycle. To organize these elements without requiring researchers to write code, the system structures every study as a

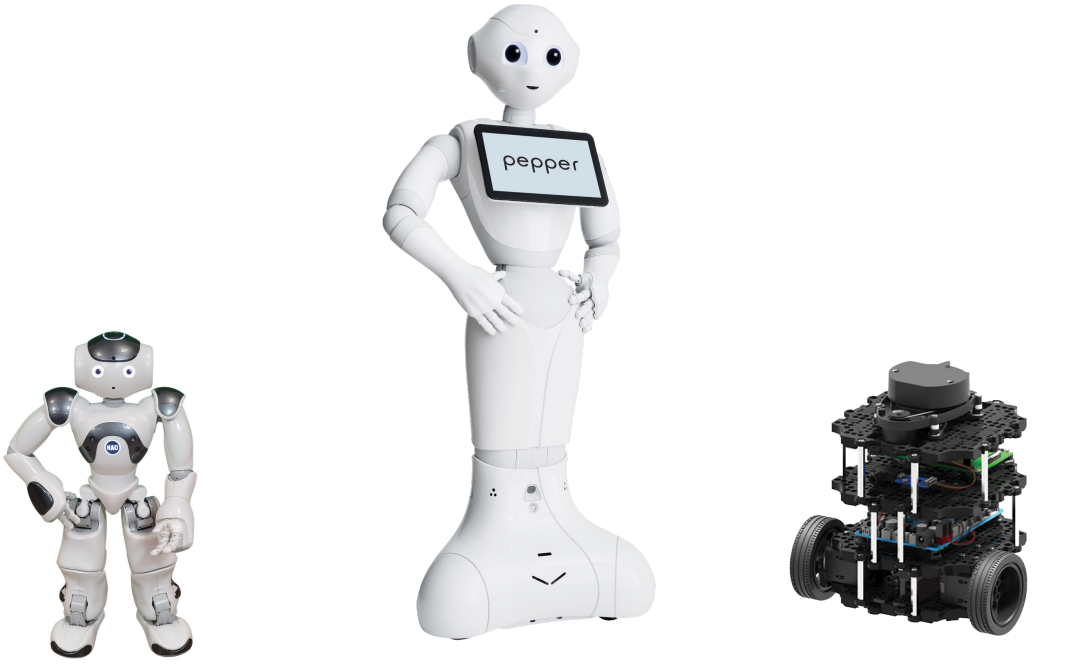
four-level hierarchy: *study* → *experiment* → *step* → *action*. This structure separates high-level protocol design from low-level execution behavior, keeping the authoring process code-free while integrating design, execution, and analysis into a single unified workflow.

I define the elements in this hierarchy as follows. A *study* is the top-level container that groups related experiments. An *experiment* is one independently runnable protocol within that study (for example, a control or experimental condition). A *step* is one phase of the protocol timeline (for example, an introduction, telling a story, or testing information recall). An *action* is the smallest executable unit inside a step (for example, trigger a gesture, play audio, or speak a prompt).

Figure 4.2 shows this hierarchical structure. Reading top-down, one study contains one or more experiments, each experiment contains one or more steps, and each step contains one or more actions.

Figure 4.3 illustrates how a protocol definition relates to its instantiation. The left column holds the protocol, defined before the study begins; the right column shows how the abstraction defined as a protocol is instantiated as independent trials. A dashed line marks the protocol/trial boundary: everything to its left was authored by the researcher before any participant arrived; everything to its right was generated during a live session. The *instantiates* arrows from the experiment node fan out to each trial record, making the relationship explicit. This separation is central to reproducibility: the same experiment specification generates a distinct, timestamped record per participant, so researchers can compare across participants without conflating what was designed with what was executed.

To illustrate the hierarchy with a concrete example, consider an interactive storytelling study with the research question: *Does how the robot tells a story affect how a human will remember the story?* The experiment might use different robots, for instance Pepper, NAO6, and TurtleBot. Figure 4.1 shows the morphology of these three different robots: Pepper and NAO6 are humanoid social robots with expressive gestures and human-like forms, while TurtleBot is a wheeled mobile robot with a visibly machine-like form and no social movement cues. In the example below, the narrative task remains the same across two robot-specific experiments; only how the robot delivers it changes.



(a) NAO6 (Humanoid) **(b)** Pepper (Social) **(c)** TurtleBot (Mechanical)

Figure 4.1: Three robot morphologies supported by the HRIStudio architecture.

Figure 4.4 maps the study presented above onto the hierarchical elements defined in Figure 4.2. The study branches into two experiments (TurtleBot with only voice, NAO6 with added gestures), each experiment uses the same sequence of ordered

steps (Intro, Story Telling, Recall Test), and each step defines the specific actions the robot will perform. The figure expands only the Story Telling step to keep the diagram readable, but Intro and Recall Test follow the same structure.

Together, these three figures motivate why the hierarchy is useful in practice. These three figures are interrelated as follows: Figure 4.2 defines the experimental structure as an abstraction; Figure 4.3 shows how the abstract experimental structure is instantiated as concrete trial records; and Figure 4.4 shows the expansion of each element of the experimental structure.

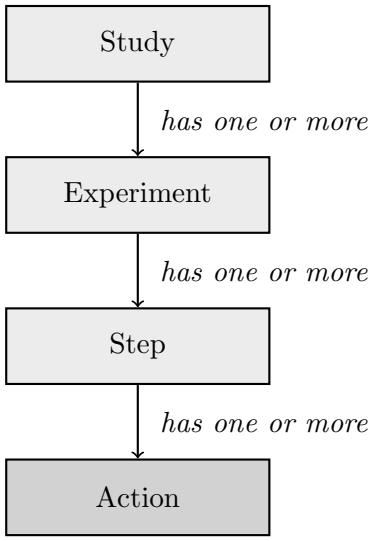


Figure 4.2: The four-level experiment specification hierarchy.

The layered structure compels researchers to define experimental protocols at multiple levels of granularity without writing code, which creates a process that is accessible to non-programmers. The step and action elements also align naturally with the sequence of events in a trial, so the wizard stays guided by the protocol while retaining control over the timing of each event, which supports the real-time control requirement (R3). Action-level execution provides a natural unit for timestamped logging and post-trial analysis, satisfying the automated logging requirement (R4).

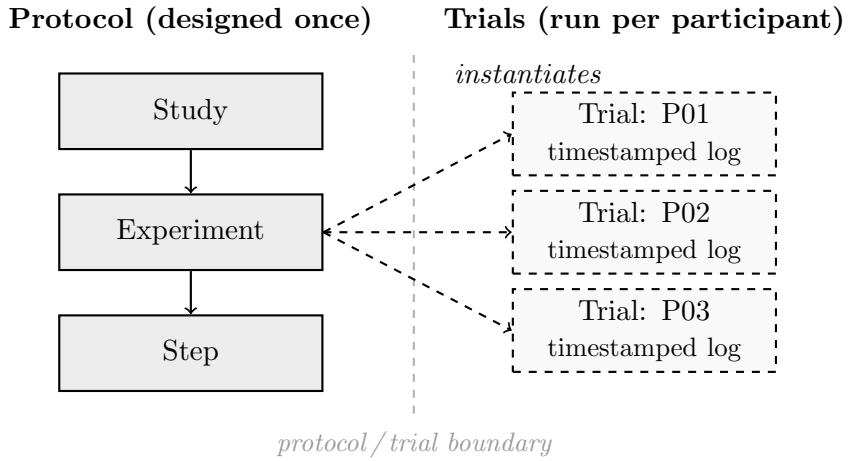


Figure 4.3: One experiment protocol instantiated as a separate trial record per participant.

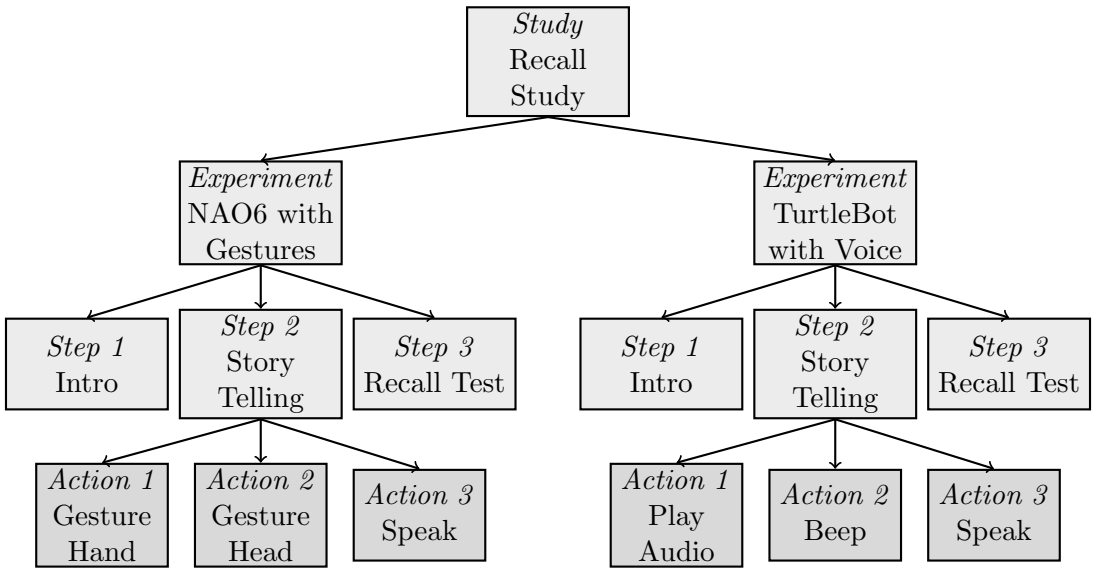


Figure 4.4: A recall study with two conditions mapped onto the four-level hierarchy.

Finally, keeping experiment definitions separate from trial instances means the same protocol can be reproduced across participants and experiments, supporting both the integrated workflow (R1) and collaborative support (R6) requirements.

4.2 Event-Driven Execution Model

To achieve real-time responsiveness while maintaining methodological rigor (R3, R5), the system uses an event-driven execution model rather than a time-driven one. In a time-driven approach, the system advances through actions on a fixed schedule regardless of what the participant is doing, so the robot might speak over a participant who is still talking, or move on before a response has been given. The event-driven model avoids this by letting the wizard trigger each action when the interaction is ready for it. Figure 4.5 contrasts the two approaches using the same four-action sequence: Greet (G), Begin Story (BS), Ask Question (AQ), and End (E). In the time-driven row, fixed intervals t_0 through t_2 define when each event fires, and dashed vertical lines show where those moments fall relative to the event-driven rows below. In both event-driven rows, the wizard fires the same four labeled events at different real-time positions (T1, a faster participant, finishes well before T2, a slower one), while both preserve the same action order.

This approach has several implications. What the event-driven model guarantees is not identical timing across trials, but consistent action ordering: every participant experiences the same sequence of protocol steps, even if the pace varies. Timing is recorded accurately, permitting researchers to analyze natural variation across participants. The wizard responds contextually without departing from the protocol; the

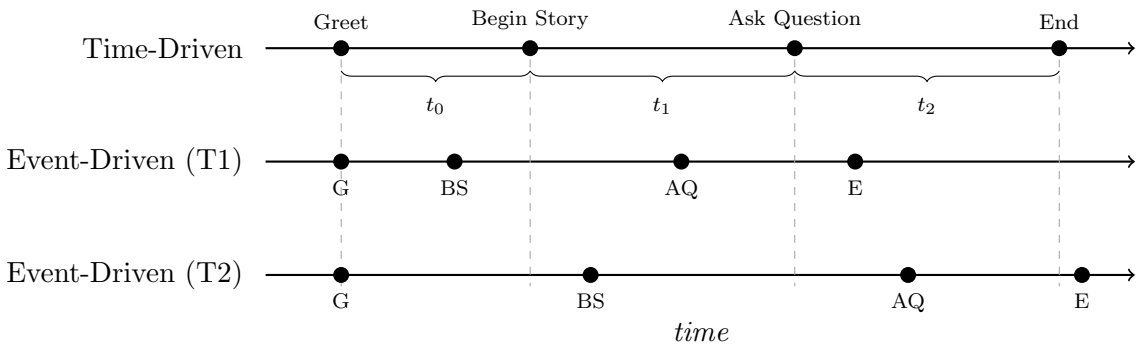


Figure 4.5: Time-driven (top) versus event-driven (bottom, two trials) execution of the same four-action protocol.

wizard remains guided by the sequence of available actions while retaining control over when to advance based on participant cues.

The system guides the wizard through the protocol step-by-step, ensuring the intended sequence is followed. Every action is logged with a timestamp whether it was scripted or not, and anything outside the protocol is flagged as a deviation. This means inconsistent wizard behavior can be evident in the data rather than disappearing into it.

4.3 Modular Interface Architecture

Researchers interact with the system through three interfaces, each one encapsulating a specific phase of an experimental study: designing a protocol, running a trial, and reviewing the results.

4.3.1 Design Interface

The *Design* interface gives researchers a drag-and-drop canvas for building experiment protocols, creating a visual programming environment. Researchers drag prebuilt action components, including robot movements, speech, wizard instructions, and conditional logic, onto the canvas and drop them into sequence. Clicking a component opens a side panel where its parameters can be set, such as the text for a speech action or the gesture name for a movement.

By treating experiment design as a visual specification task, the interface lowers technical barriers (R2). Researchers can assemble interaction logic by dragging components into sequence and setting parameters naturally, without even having to write code. The resulting protocol specification is also human-readable and shareable alongside experimental results. The specification is stored in a structured format that can be displayed as a timeline for analysis and executed directly by the platform's runtime. This property is central to reproducibility: a third party with access to the specification can run the experiment faithfully without reverse-engineering the original system.

4.3.2 Execution Interface

During trials, the *Execution* interface keeps the wizard informed of exactly where they are in the protocol. The current step, the available actions, and the robot's current state are all updated in real time as the trial progresses.

The *Execution* interface also exposes a set of manual controls for actions that

fall outside the scripted protocol. A *deviation* is a spontaneous action introduced by the wizard in response to a reaction of the human subject that was not anticipated when the script was created. Consider a human subject who asks an unexpected question mid-trial: the wizard can trigger an unscripted speech response on the spot rather than leaving the interaction to stall, keeping the interaction feeling natural for the human subject. Critically, the system does not ignore these deviations from the script. Every deviation is timestamped and written to the trial log, giving researchers a complete picture of what actually happened versus what was planned. This makes unscripted actions a feature rather than a source of noise: the wizard retains real-time control over the interaction, and the logging infrastructure captures everything needed for post-trial analysis.

Additional researchers can simultaneously access a live view of a trial through the platform's Dashboard by selecting a trial to "spectate." Multiple researchers observing the same trial view an identical synchronized display of the wizard's controls, human subject interactions, and robot state, supporting real-time collaboration and interdisciplinary observation (R6). Observers can take notes and mark significant moments without interfering with the wizard's control or the human subject's experience.

4.3.3 Analysis Interface

After a trial concludes, the *Analysis* interface lets researchers review everything that was recorded: video of the interaction, audio, timestamped action logs, and robot sensor data, all scrubable from a single timeline. Researchers can annotate significant moments and export segments for further analysis. Because the same platform

produced both the protocol and the recording, the interface eliminates the need for manual cross-referencing by showing exactly where the execution matched the design and where it deviated.

4.4 Data Flow and Infrastructure Implementation

To ensure that data from every experimental phase remains traceable, the system organizes its internals into three architectural layers and defines a clear data pathway from protocol design through post-trial analysis, covering how experiment specifications, control commands, and recorded data move through the system.

4.4.1 Architectural Layers

HRIStudio separates its communicative and functional responsibilities into distinct layers, in a manner analogous to the layered reference models used in networking software. More specifically, the system is organized as a three-layer architecture, as shown in Figure 4.6, each layer with a specific responsibility:

User Interface layer. Runs in researchers' web browsers and exposes the three interfaces (Design, Execution, Analysis), managing user interactions such as clicking buttons, dragging and dropping experiment components, and reviewing experimental results.

Application Logic layer. Operates as a server process that manages experiment data, coordinates trial execution, authenticates users, and orchestrates commu-

nication between the interface and the robot.

Data and Robot Control layer. Encompasses long-term storage of experiment protocols and trial data, as well as direct communication with robot hardware.

This separation of concerns provides two concrete benefits. First, each layer can evolve independently: improving the user interface requires no changes to robot control logic, and swapping in a different storage backend requires no changes to the execution engine. Second, the separation enforces clear responsibilities: the user interface never directly commands robot hardware; all robot actions flow through the application logic layer, which maintains consistent logging. Figure 4.6 shows that HRIStudio separates interface behavior, execution logic, and robot/data operations into distinct layers with explicit boundaries.

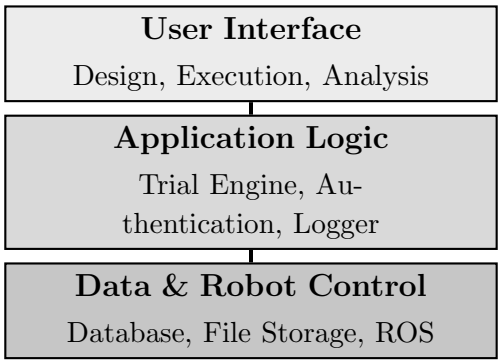


Figure 4.6: Three-layer architecture separates user interface, application logic, and data/robot control.

4.4.2 Data Flow Through Experimental Phases

During the design phase, researchers create experiment specifications that are stored in the system database. During a trial, the system manages bidirectional communication between the wizard's interface and the robot control layer. All actions, sensor

data, and events are streamed to a data logging service that stores complete records. After the trial, researchers can inspect these records through the Analysis interface.

The flow of data during a trial proceeds through six distinct phases as discussed below; these phases are summarized in Figure 4.7:

1. A researcher creates an experiment protocol using the Design interface.
2. When a trial begins, the application server loads the protocol and allows the wizard to step through it, sending commands to the robot and waiting for events such as wizard inputs, sensor readings, or timeouts.
3. Every action, both planned protocol steps and deviations, is immediately written to the trial log with precise timing information.
4. The *Execution* interface continuously displays the current state, allowing the wizard and observers to monitor the progress of a trial in real-time.
5. When the trial concludes, all recorded media (video and audio) is transferred from the browser to the server and persisted in a database as part of the trial record.
6. The *Analysis* interface retrieves the stored trial data and reconstructs exactly what happened, synchronizing notable events with the video and audio recordings.

This design creates automatically a comprehensive documentation of every trial, supporting both fine-grained analysis and reproducibility. Researchers can review not just what they intended to happen, but what actually did happen, including timing variations and deviations.

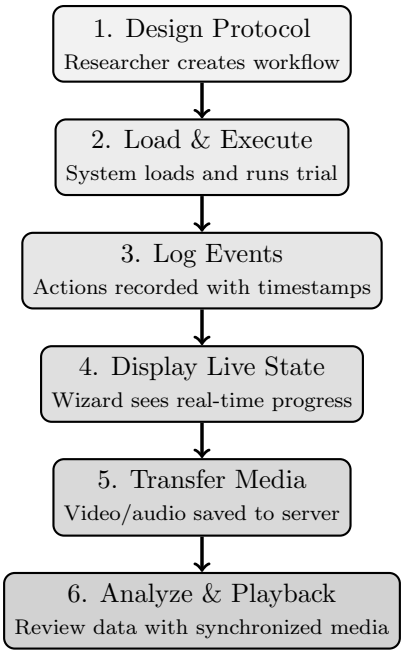


Figure 4.7: Six-phase trial data flow.

4.4.3 Requirements Satisfaction

The design choices described in this chapter were made to meet the requirements from Chapter 2. Having the researcher work through a single platform from protocol creation to post-trial review satisfies R1 (integrated workflow: design, execution, and analysis in one environment) without extra tooling. The visual drag-and-drop Design interface removes the need for programming knowledge, satisfying R2 (low technical barriers) by keeping the system accessible to researchers without a software background. Event-driven execution satisfies R3 (real-time control) by giving the wizard control over pacing while keeping the trial on protocol. All actions are logged automatically at the system level, satisfying R4 (automated logging) without requiring researchers to add logging by hand. The three-layer architecture decouples action specifications from robot-specific commands, satisfying R5 (platform agnosticism) by letting the same protocol run on different hardware without modification.

Finally, shared live views and multi-user access let interdisciplinary teams observe and annotate the same trial simultaneously, satisfying R6 (collaborative support).

4.5 Chapter Summary

This chapter described the architectural design with emphasis on how each design choice directly implements the infrastructure requirements identified in Chapter 2. The hierarchical organization of experiment specifications enables intuitive, executable design. The event-driven execution model balances protocol consistency with realistic interaction dynamics. The modular interface architecture separates concerns across design, execution, and analysis phases while maintaining data coherence. The integrated data flow ensures that reproducibility is supported by design rather than by afterthought. The following chapter presents HRIStudio, the platform built on these design principles, describing the specific technologies and architectural components that bring them to life.

Chapter 5

Implementation

HRIStudio is a complete, operational platform that realizes the design principles established in Chapter 4. As the primary artifact of this thesis, it demonstrates that those principles are not merely theoretical: the hierarchical specification model, the event-driven execution model, and the integrated data flow can be built into a system that real researchers use without programming expertise. Any system built on those principles could satisfy the same requirements; HRIStudio is the implementation that proves they work in practice. This chapter explains how HRIStudio realizes those principles, covering the architectural choices and mechanisms behind how the platform stores experiments, executes trials, integrates robot hardware, and controls access. The specific technologies used are presented in Appendix C.

5.1 Platform Architecture

HRIStudio follows the model of a web application. Users access it through a standard browser without installing specialized software, and the entire study team, including researchers, wizards, and observers, connect to the same shared system. This eliminates the need for a local installation and ensures the platform works identically on any operating system, directly addressing the low-technical-barrier requirement (R2, from Chapter 2). It also enables easy collaboration (R6): multiple team members can access experiment data and observe trials simultaneously from different machines without any additional configuration.

I organized the system into three layers: User Interface, Application Logic, and Data & Robot Control. This layered structure is presented in Chapter 4 and shown in Figure 4.6. In practice, the User Interface layer runs in each researcher's browser (the client), while the Application Logic and Data & Robot Control layers run on a shared application server.

While the system can run entirely on a single machine for local testing, this architecture allows the components to be distributed across different systems. The application server can be hosted centrally or even in a remote data center, enabling observers to connect to a live trial from any location with internet access. In such a configuration, it is essential that the robot control hardware and the client computer running the wizard's Execution interface stay on the same local network as the robot. This ensures that the WebSocket-based communication between the wizard and the robot bridge maintains low latency, as a noticeable delay between the wizard's input and the robot's response would break the interaction.

This flexibility of deployment also addresses the varying data security and compliance needs of different research institutions. A lab may choose to host HRIStudio on a public-facing server to prioritize collaborative ease and accessibility for remote team members. Alternatively, a lab with strict data privacy requirements or institutional review board (IRB) constraints can deploy the entire stack on a private, air-gapped network. Because the platform is self-contained and does not rely on external cloud services for its core execution logic, researchers have full control over where their experimental data is stored and who can access it.

I implemented all three layers in the same language: TypeScript [16], a statically-typed superset of JavaScript. The single-language decision keeps the type system consistent across the full stack. When the structure of experiment data changes, the type checker surfaces inconsistencies across the entire codebase at compile time rather than allowing them to appear as runtime failures during a trial.

HRIStudio is released as open-source software under the MIT License, with the application hosted at a public repository [17]. The companion robot plugin repository [18] is maintained as a git submodule and is updated whenever HRIStudio requires schema or protocol updates. Both repositories are available for inspection, extension, and deployment by other research groups.

HRIStudio is implemented as a set of containerized services that work together to provide the platform's functionality. This modular architecture ensures that each component can be scaled or replaced independently as requirements change.

The HRIStudio system consists of three primary services: a Next.js application server that handles the user interface and business logic, a PostgreSQL database for

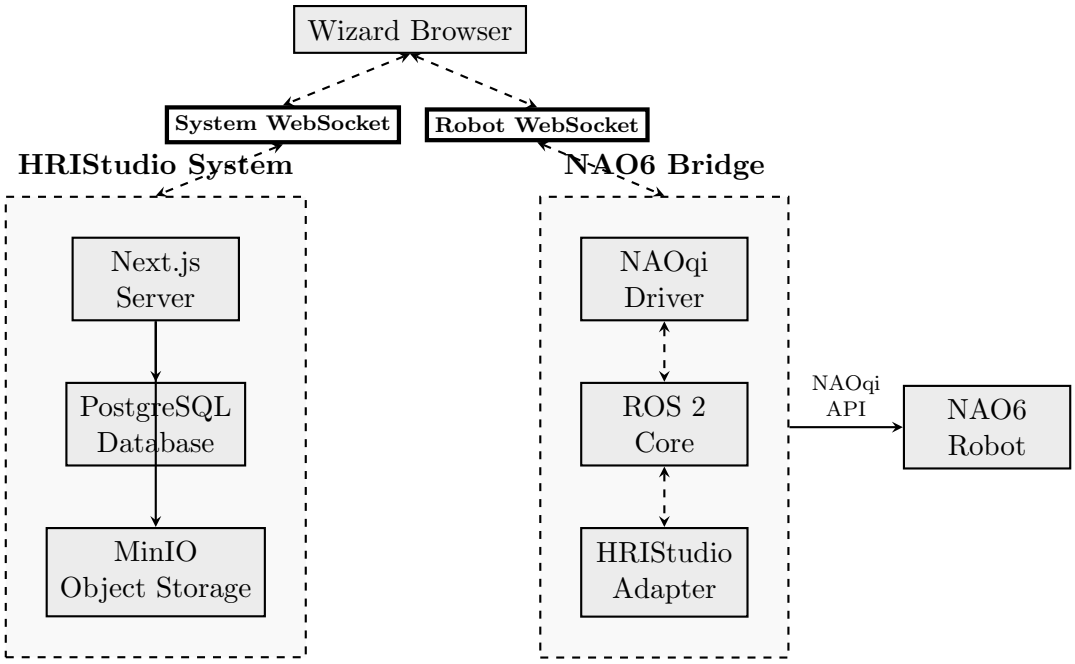


Figure 5.1: Containerized HRIStudio and NAO6 integration architecture.

persistent storage of experiment and trial data, and a MinIO object storage service for managing large media files like video and audio recordings. For robot integration, the `nao6-hristudio-integration` bridge also employs a containerized structure consisting of the NAOqi driver, a ROS 2 core for message routing, and a specialized adapter that communicates with HRIStudio.

During a live trial, the wizard's browser establishes two independent WebSocket connections. The System WebSocket connects to the HRIStudio server to manage trial state, protocol progression, and logging. The Robot WebSocket connects directly to the integration bridge to provide low-latency control of the robot platform. This split-connection model ensures that system-level management does not introduce latency into the robot's physical responses.

5.1.1 Working with AI Coding Assistants

The scale of the implementation described in this chapter, a full-stack TypeScript application spanning user interface, application logic, persistent storage, and real-time robot control, would not have been possible within the timeframe of this thesis without the use of AI coding assistants. I distinguish clearly between the engineering and implementation roles in this work: I architected the system, made the design decisions documented in Chapter 4 and this chapter, specified the behavior and constraints of each component, and reviewed and integrated all code before it entered the codebase. AI agents acted as software developers working under that direction, producing TypeScript code in response to the specifications I provided and the feedback I gave as the implementation evolved. The division of labor was consistent throughout: I engineered, they implemented.

The tools I used in this capacity spanned several vendors and interaction paradigms, and the set evolved as the AI landscape changed over the course of the project. Claude [19] was the conversational model I relied on most consistently for design discussions and code review. I used Claude Code [20], OpenCode [21], the Gemini CLI [22], and Google Antigravity [23] as terminal- and editor-integrated coding agents for implementing the features I specified; the Zed editor [24] served as the surrounding development environment and provided its own AI-assisted editing features. These tools overlapped in places, but I generally used one at a time and switched between them as new capabilities became available and as I learned which tool suited which kind of work. Appendix D documents this workflow in more detail: the division of responsibility between me and the agents, the kinds of tasks each category of tool handled well, and the limits I ran into.

5.2 Experiment Storage and Trial Logging

The system saves experiment descriptions to persistent storage when a researcher completes them in the Design interface. A saved experiment is a complete, reusable specification that a researcher can run across any number of trials without modification.

When a trial begins, the system creates a new trial record linked to that experiment. The system writes every action the wizard triggers to that record with a precise timestamp, whether scripted or not, including any unscripted actions triggered outside the protocol. The system flags those unscripted actions as deviations. The Execution interface records video, audio, and robot sensor data alongside the action log for the duration of the trial. The Analysis interface can directly compare what was planned against what was executed for any trial, without any manual work by the researcher, because the trial record and the experiment reference the same underlying specification. Figure 5.2 shows the structure of a completed trial record: action log entries, video, audio, and robot sensor data all share a common timestamp reference so the Analysis interface can align them without manual synchronization; dashed lines mark step boundaries; and the system flags any deviation from the experiment specification at the appropriate position in the timeline.

Video and audio are recorded locally in the wizard's browser during the trial rather than streamed to the server in real time. The wizard's browser is the canonical recording client because the wizard is the only role required for a trial to run; observer and researcher roles connect in read-only capacities and do not capture media. Recording locally prevents network delays or server load from dropping frames or degrading

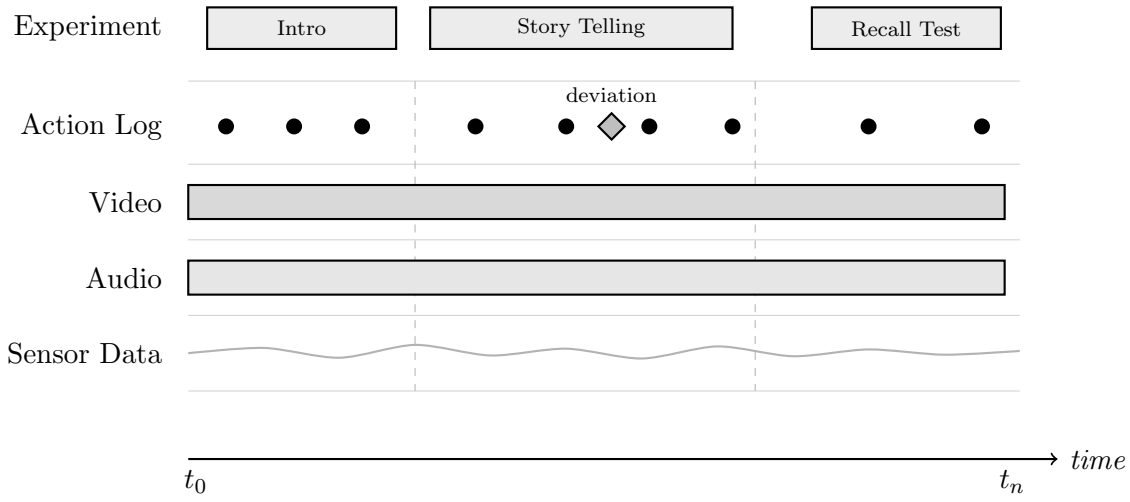


Figure 5.2: Structure of a completed trial record, showing synchronized action log, media, and sensor tracks.

audio quality during the interaction. When the trial concludes, the wizard's browser transfers the complete recordings to the server and associates them with the trial record. The Analysis interface can align video and audio with the logged actions without any manual synchronization, because the timestamp when recording starts is logged alongside the action log.

The system stores structured and media data separately. Experiment specifications and trial records are stored in the same structured database, which makes it efficient to query across trials (for example, retrieving all trials for a specific participant or comparing action timing across conditions). Video and audio files are stored in a dedicated file store, since their size makes them unsuitable for a database and the system never queries their content directly.

Figure 5.3 shows the Analysis interface reconstructing a completed trial. The recorded video is presented alongside a synchronized action log, with each logged event linked to its moment in the recording so researchers can jump directly to the

corresponding interaction without manual cross-referencing.

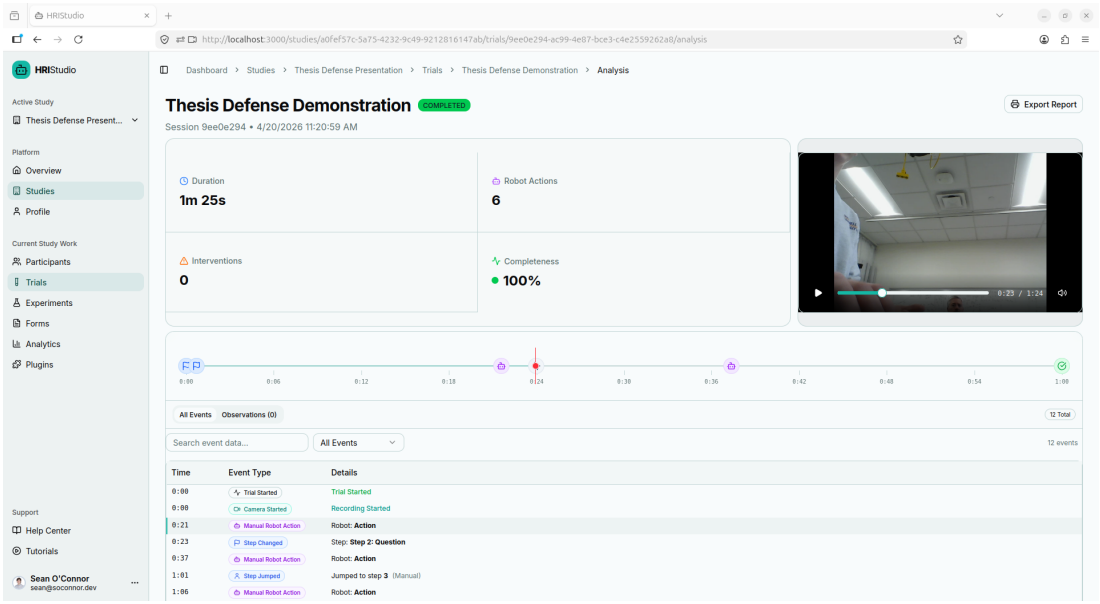


Figure 5.3: The HRIStudio Analysis interface showing a completed trial with video and a synchronized, timestamped action log.

5.3 The Execution Engine

The execution engine is the component that runs a trial: it loads the experiment, manages the wizard's connection, sends robot commands, and keeps all connected clients in sync.

When a trial begins, the server loads the experiment and maintains a live connection to the wizard's browser and any observer connections. The wizard controls how time advances from action to action. When the wizard triggers an action, the server sends the related command to the robot, writes the log entry, and pushes the updated experiment state to all connected clients in the same operation, keeping the wizard's view, the observer view, and the actual robot state synchronized in real time.

No two human subjects respond identically to an experimental protocol. One subject gives a one-word answer; another offers a paragraph; a third asks the robot a question the script never anticipated. Unscripted actions give the wizard the tools to record how these interactions unfold when deviations from the script are required. The wizard triggers them via the manual controls in the Execution interface, the robot command runs, and the system logs the action with a deviation flag. This design preserves research value: the interaction gains the flexibility only a human can provide, and that flexibility appears explicitly in the record rather than disappearing into it.

Figure 5.4 shows the Execution interface as it appears to a wizard during a live trial. The current step is highlighted in the protocol sidebar, the available actions for that step are surfaced as triggerable buttons, and the wizard has manual-control affordances for introducing unscripted actions that the system will flag as deviations in the trial log.

5.4 Robot Integration

A plugin file describes each robot platform, listing the actions it supports and specifying how each one maps to a command the robot understands. The execution engine reads this file at startup and uses it whenever it needs to dispatch a command: it looks up the action type, assembles the appropriate message, and sends it to the robot over a bridge process running on the local network. For the NAO6 platform, I developed a specialized ROS-based bridge called `nao6-hristudio-integration` [25] that translates HRIStudio commands into the NAOqi API calls required by the robot.

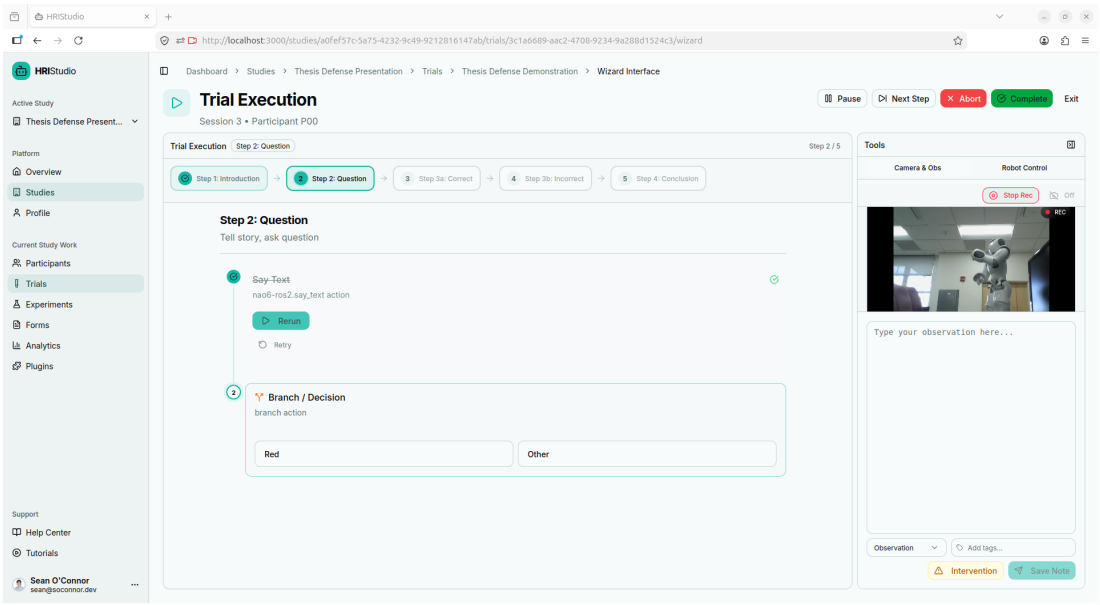


Figure 5.4: The HRIStudio Execution interface during a live trial, showing the current step, available actions, and manual deviation controls.

The web server itself has no knowledge of any specific robot; all hardware-specific logic lives in the plugin file.

The execution engine treats control flow elements such as branches and conditionals, which function as elements of a computer program, the same way as robot actions. These control-flow elements appear as action groups in the experiment and are evaluated during the trial, so researchers can freely mix logical decisions and physical robot behaviors when designing an experiment without any special handling.

Figure 5.5 illustrates this mapping using NAO6 and TurtleBot as an example. Actions a platform does not support (such as `raise_arm` on TurtleBot) appear as explicitly unsupported in the plugin file rather than silently failing. Because all hardware-specific logic lives in the plugin file, the experiment itself does not change between platforms.

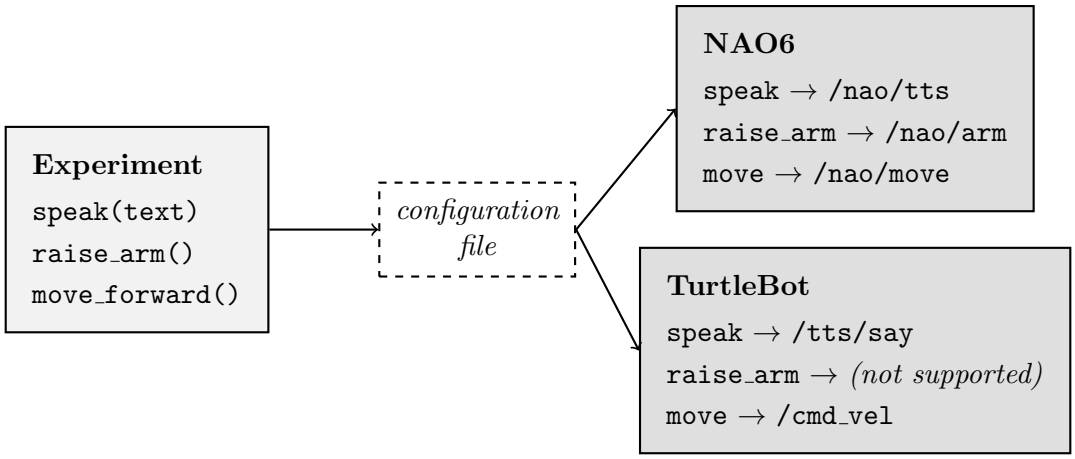


Figure 5.5: Abstract experiment actions translated to platform-specific robot commands through per-platform plugin files.

5.4.1 Containerized Development Environment

To support development and testing for the NAO platform, I also developed `nao-workspace`, a containerized workspace [26]. This was motivated by the technical constraints of Choregraphe and its related libraries, which only supported x86-64 systems running Ubuntu 22.04. The containerized structure was the only way I could run the proprietary NAO development tools on modern hardware. While I developed this stack primarily to enable technical testing and material preparation during the project, the resulting tooling may be useful to other HRI researchers facing similar platform constraints.

5.5 Access Control

I implemented access control using a role-based access control (RBAC) model with two layers. System-level roles govern what a user can do across the platform (ad-

ministrator, researcher, wizard, observer), while study-level roles govern what a user can see and do within a specific study (owner, researcher, wizard, observer). The two layers are checked independently, so a user who is a wizard on one study can be an observer on another without any additional configuration. Within a study, the four study-level roles define a clear separation of capabilities: those who own the study, those who design it, those who run it, and those who observe it. This enforces need-to-know access at the study level so that each team member sees or is able to modify only what their role requires. The capabilities and constraints for each role are described below:

Owner. Full control over the study: can invite or remove members, configure the study settings, and access all data.

Researcher. Can create and modify experiment designs and review all collected trial data, but cannot manage team membership.

Wizard. Can trigger actions during a trial and view the execution interface, but cannot modify the experiment design or access other wizards' sessions.

Observer. Read-only access: can watch a trial in real time and annotate significant moments, but cannot trigger actions or modify any data.

The role definitions above determine who can view and change data during normal study operation. The role system also supports what is known as a double-blind design [1], where neither the wizard nor the researcher has access to condition assignments or results until the study concludes. For example, the Owner can restrict a Wizard's view of which condition a human subject has been assigned to, and can

prevent Researchers from accessing result data until all trials are complete, without any changes to the underlying experiment.

5.6 Architectural Challenges

The following two problems required specific solutions during implementation.

Execution latency. During a trial, the execution engine must respond quickly to wizard input, as a noticeable delay between the button press and the robot's action can disrupt the interaction. I addressed this by maintaining a persistent network connection to the robot bridge for the duration of each trial. The connection is established once at trial start and kept open, eliminating per-action setup overhead.

Multi-source synchronization. The Analysis interface requires aligning data streams captured at different sampling rates by different components: video, audio, action logs, and sensor data. The solution is a shared time reference: every data source records its timestamps relative to the same trial start time, t_0, so the Analysis interface can align all tracks without requiring manual calibration.

5.7 Implementation Status

HRIStudio is fully operational for controlled Wizard-of-Oz studies. The Design, Execution, and Analysis interfaces are complete and integrated with one another. The

execution engine handles scripted and unscripted actions with full timestamped logging, and I validated robot communication on the NAO6 platform during development. A researcher can design an experiment, run a live trial with a wizard, and review the resulting logs and recordings without modification to the platform's core architecture or execution workflow.

Work remaining for future development includes broader validation of the plugin file approach on robot platforms beyond NAO6, as discussed further in Chapter 9.

5.8 Chapter Summary

This chapter described how HRIStudio realizes the design principles from Chapter 4 in practice. Experiments are persistent, reusable specifications that produce complete, comparable trial records. The execution engine is event-driven rather than timer-driven, keeping the wizard in control of pacing while logging every action automatically. Per-platform plugin files keep the execution engine hardware-agnostic. The role system enforces access control at the study level. The platform is fully operational for controlled WoZ studies today, demonstrated through the pilot validation study presented in Chapter 6. The design principles are general; HRIStudio shows they are workable.

Chapter 6

Pilot Validation Study

This chapter presents the pilot validation study used to evaluate whether HRIStudio improves accessibility and reproducibility in WoZ-based HRI research. It defines the research questions, study design, task, apparatus, procedure, and measurement instruments.

6.1 Research Questions

The validation study targets the two problems established in Chapter 2. The first is the *Accessibility Problem*: existing tools require substantial programming expertise, which prevents domain experts from conducting independent HRI studies. The second is the *Reproducibility Problem*: without structured logging and protocol enforcement, experiment execution varies across participants and wizards in ways that are difficult to detect or control after the fact.

These problems give rise to two research questions. The first is whether HRIStudio enables domain experts without prior robotics experience to successfully implement a robot interaction from a written specification. The second is whether HRIStudio produces more reliable execution of that interaction compared to standard practice.

I hypothesized that HRIStudio would improve both accessibility and reproducibility compared to Choregraphe: wizards using HRIStudio would more completely and correctly implement the written specification, and their designs would execute more reliably during the trial.

6.2 Study Design

I used what Bartneck et al. [1] call a *between-subjects design*, in which each participant is assigned to only one condition. To ensure that programming experience was balanced across conditions, I stratified assignment by self-reported programming background: each wizard was first classified as having *None*, *Moderate*, or *Extensive* programming experience, and then randomly assigned within that stratum to HRIStudio or Choregraphe. This produced a design in which each condition contained exactly one wizard at each experience level, reducing the risk that tool effects would be confused with differences in programming experience. Both groups received the same task, the same time allocation, and a similar training structure. Because each wizard used only one tool, the design also avoided carryover effects from prior exposure to the other condition.

6.3 Participants

Wizards. A primary claim of HRIStudio is that it lowers the technical barrier for domain experts who are not programmers; testing this claim with its intended user population was therefore a primary goal of participant recruitment. I recruited six Bucknell University faculty members drawn from across departments to serve as wizards, deliberately targeting both ends of the programming experience spectrum: those with substantial programming backgrounds as well as those who described themselves as non-programmers or having minimal coding experience. Drawing wizards from outside computer science allows the data to speak to whether that claim holds for the intended user population.

The key inclusion criterion for all wizards was no prior experience with either the NAO robot or Choregraphe software specifically. This controls for tool familiarity so that performance differences reflect the tools themselves rather than prior exposure. I recruited wizards through direct email, and participation was framed as a voluntary software evaluation unrelated to any professional obligations.

Sample size rationale. I chose to recruit six wizard participants ($N = 6$), believing that this sample size is appropriate for a pilot validation study whose goal is directional evidence and failure-mode identification rather than effect-size estimation for a broad population. This scale is consistent with pilot and feasibility studies in HRI, where small N designs are common in early-stage tool validation [27]. Findings should be interpreted as preliminary evidence and directional indicators rather than as conclusive substantiation of any claims.

6.4 Task

The task chosen was to have a robot tell a story to a human subject and later evaluate if that subject could recall a specific detail.

Both wizard groups received the same written task specification: the *Interactive Storyteller* scenario. The specification described a robot that introduces an astronaut named Kai, narrates her discovery of a red rock on Mars, asks a recall question, and delivers a response according to the answer given. The full specification, including exact robot speech, required gestures, and branching logic, is reproduced in Appendix A. This scenario is representative of HRI tasks in which a robot conveys information to a human subject; one might, for example, measure whether a robot or human storyteller produces better recall in subjects.

This scenario was chosen because it requires several distinct capabilities: speech actions, gesture coordination, conditional branching, and a defined conclusion. In both conditions, wizards had to translate the same written protocol into an executable interaction script, including action ordering, branching logic, and timing decisions. In Choregraphe, that meant assembling and connecting behavior nodes in a finite state machine. In HRIStudio, it meant building a sequential action timeline with conditional branches. This makes the task a direct comparison of how each tool supports coding the robot behavior required by the same protocol.

6.5 Robot Platform and Software Apparatus

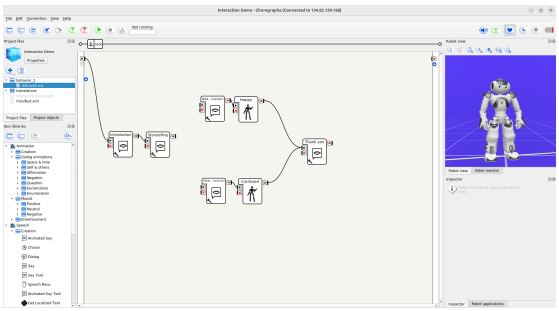
Both conditions used the same NAO humanoid robot (Figure 6.1), a platform approximately 0.58 meters tall capable of speech synthesis, animated gestures, and head movement. Using the same hardware ensured that any differences in execution quality were attributable to the software, not the robot.

Figure 6.1: The NAO6 humanoid robot used in both conditions of the pilot study.

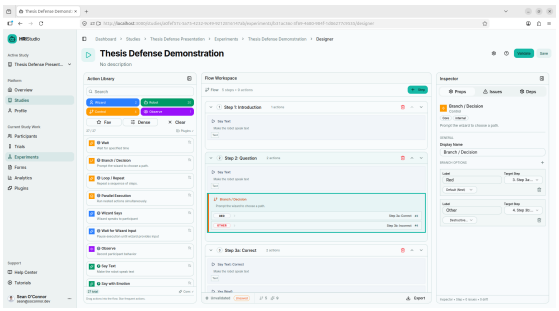
The control condition used Choregraphe [10], a proprietary visual programming tool developed by Aldebaran Robotics and the standard software for NAO programming. Choregraphe organizes behavior as a finite state machine: nodes represent states and edges represent transitions triggered by conditions or timers.

The experimental condition used HRIStudio, described in Chapter 5. HRIStudio organizes behavior as a sequential action timeline with support for conditional branches. Unlike Choregraphe, it abstracts robot-specific commands through plugin files, though for this study both tools controlled the same NAO platform.

Figure 6.2 places the two design environments side by side. On the left, Choregraphe's behavior-box canvas (Figure 6.2a) lets the wizard wire nodes and transitions in a finite-state-machine layout. On the right, HRIStudio's experiment designer (Figure 6.2b) presents the same protocol as a vertical action timeline with dedicated blocks for speech, gesture, and conditional branching.



(a) Choregraphe: behavior-box canvas with nodes and transitions.



(b) HRIStudio: vertical action timeline with structured step and action blocks.

Figure 6.2: The two design environments compared. Each wizard used one of these tools to implement the Interactive Storyteller specification.

6.6 Procedure

Each wizard completed a single 60-minute session structured in four phases.

6.6.1 Phase 1: Training (15 minutes)

I opened each session with a standardized tutorial tailored to the wizard's assigned tool. The tutorial covered how to create speech actions, specify gestures, define conditional branches, and save the completed design. Training was intentionally allocated 15 minutes to allow enough time for wizards to ask clarifying questions about

the tool before the design challenge began, while still simulating first encounter with a new tool without extensive onboarding. I answered clarification questions during this phase but did not offer hints about the design challenge.

6.6.2 Phase 2: Design Challenge (30 minutes)

The wizard received the specification and had 30 minutes to implement it using their assigned tool. Using a structured observer data sheet (found in Appendix A), I logged every instance in which I provided assistance to the wizard, categorizing each by type: *tool-operation* (T), *task clarification* (C), *hardware or technical* (H), or *general* (G). For each tool-operation intervention, I also recorded which rubric item it pertained to. If the wizard declared completion before the time limit, the remaining time was used to review and refine the design.

6.6.3 Phase 3: Live Trial (10 minutes)

After the design phase, the wizard ran their completed program to execute the designed interaction on the robot. I continued logging any researcher interventions during the trial using the same type categories, noting the relevant ERS rubric item for any tool-operation intervention.

6.6.4 Phase 4: Debrief (5 minutes)

Following the trial, the wizard completed the System Usability Scale survey (found in Appendix A). The DFS and ERS were scored during and immediately after the session using live observation and the Observer Data Sheet.

6.7 Measures

The study collected five measures, two primary and three supplementary, operationalized through five instruments. They are described as follows.

6.7.1 Design Fidelity Score

I define the Design Fidelity Score (DFS) as a measure of how completely and correctly the wizard implemented the specification. I evaluated the exported project file against nine weighted criteria grouped into three categories: speech actions, gestures and actions, and control flow and logic. Each criterion is scored as present, correct, and independently achieved.

The DFS rubric includes an *Assisted* column. For each rubric item, I marked a T if I provided a tool-operation intervention specifically for that item during the design phase (for example, if I explained how to add a gesture node or how to wire a conditional branch). T marks are recorded and reported separately alongside the DFS score; they do not affect the Points total. This preserves the DFS as a clean measure of design fidelity while providing a parallel record of where tool-specific assistance

was needed. General interventions (task clarification, hardware issues, or momentary forgetfulness) are not marked T, because those categories of difficulty are independent of the tool under evaluation.

DFS is motivated by a gap identified by Riek [14], whose systematic review of 54 published WoZ studies found that only 11% constrained wizard behavior and fewer than 6% described wizard training procedures. Porfirio et al. [28] similarly argued that formal, verifiable behavior specifications are a prerequisite for reproducible HRI. The DFS applies these recommendations as a weighted rubric scored against the exported project file. The complete rubric is reproduced in Appendix A. This measure addresses the question: did the tool allow a wizard to independently produce a correct design?

6.7.2 Execution Reliability Score

I define the Execution Reliability Score (ERS) as a measure of whether the designed interaction executed as intended during the live trial. I scored the ERS live and immediately after the session, using the Observer Data Sheet and the wizard's exported project file. Evaluation criteria included whether the robot delivered the correct speech at each step, whether gestures executed and synchronized with speech, whether the conditional branch was present in the design and executed during the trial, and whether any errors, disconnections, or hangs occurred.

The ERS rubric applies the same *Assisted* modifier as the DFS, extended to the trial phase. Any tool-operation intervention I provided during the trial (for example, explaining to the wizard how to launch or advance their program) caps the affected

ERS item at half points. This is scored separately from design-phase interventions: a wizard who needed help only during design can still achieve a full ERS score if the trial runs without assistance, and vice versa. The rubric records whether the trial reached its conclusion step. I additionally note whether any branch resolved through programmed conditional logic or through manual intervention by the wizard during execution.

This measure responds directly to Riek's [14] finding that only 3.7% of published WoZ studies reported any measure of wizard error, making it nearly impossible to determine whether execution matched design intent [3, 4]. The complete rubric is reproduced in Appendix A. This measure addresses the question: did the design translate reliably into execution without researcher support?

6.7.3 System Usability Scale

The System Usability Scale (SUS) is a validated 10-item questionnaire measuring perceived usability, created by Brooke [29]. Wizards completed the SUS after the debrief phase. Scores range from 0 to 100, with higher scores indicating better perceived usability. The full questionnaire is reproduced in Appendix A.

6.7.4 Intervention Log and Session Timing

During each session, I maintained a structured intervention log on the observer data sheet, recording the timestamp, type code, affected rubric item number, and a brief description for every instance in which I assisted the wizard. The four intervention

type codes are:

T (tool-operation). I explained how to operate a specific feature of the assigned software tool.

C (task clarification). I clarified the written specification or an aspect of the task design.

H (hardware or technical). I addressed a robot connection issue or other technical problem outside the wizard's control.

G (general). Brief assistance not attributable to the tool or the task, such as momentary forgetfulness.

Only T-type interventions affect rubric scoring; the others are recorded to provide context for interpreting session flow and wizard experience. I also recorded the actual duration of each session phase and the time at which the wizard completed or abandoned the design, providing supplementary evidence about tool accessibility beyond the DFS score itself.

6.8 Measurement Instruments

The five measures are designed to work together. The DFS and ERS address separate phases of the session: DFS captures what was designed, and ERS captures whether that design translated faithfully into execution. Taken together, they make it possible to distinguish a wizard who implemented the specification correctly but whose design failed during the trial from one whose design was incomplete but executed without

researcher assistance. The SUS grounds both scores in the wizard's subjective experience of the tool. The intervention log and session timing are supplementary: they do not directly answer the research questions but provide context for interpreting the primary scores, particularly for understanding whether help requests concerned the tool itself or the task.

Table 6.1 summarizes the five instruments, when they were collected, and which research question each addresses.

Instrument	**What it captures**	**When collected**	**Research question**
Design Fidelity Score (DFS)	Completeness and correctness of the wizard's implementation; caps items where tool-operation assistance was given	Post-session file review	Accessibility
Execution Reliability Score (ERS)	Whether the interaction executed as designed during the trial; caps items where trial-phase tool assistance occurred	Live and post-trial (ODS)	Reproducibility
System Usability Scale (SUS)	Wizard's perceived usability of the assigned tool	Debrief phase	User experience
Intervention Log	Timestamped record of all researcher assistance by type (T/C/H/G) and affected rubric item	Throughout session	Supplementary
Session Timing	Actual duration of each phase; time to design completion	Throughout session	Supplementary

Table 6.1: Measurement instruments used in the pilot validation study.

6.9 Chapter Summary

This chapter described the structure of a pilot between-subjects study I designed to test whether the design principles formalized in Chapters 4 and 5 produce mea-

surably different outcomes from existing practice. Six wizard participants ($N = 6$), drawn from across departments and spanning the programming experience spectrum, each designed and ran the Interactive Storyteller task on a NAO robot using either HRIStudio or Choregraphe. Each 60-minute session was structured in four phases: a 15-minute standardized tutorial, a 30-minute design challenge, a 10-minute live trial, and a 5-minute debrief. I measured design fidelity (DFS) and execution reliability (ERS) against the written specification, applying a per-item scoring modifier that caps any rubric criterion for which tool-operation assistance was given. I also collected perceived usability via the SUS, a structured intervention log categorizing all researcher assistance by type, and session phase timings. Chapter 7 presents the results.

Chapter 7

Results

This chapter presents the results of the pilot validation study described in Chapter 6. Because this is a small pilot, I report descriptive statistics and qualitative observations rather than inferential tests. The goal is directional evidence: the chapter reports whether patterns in the data consistently favor HRIStudio across the primary and supplementary measures.

7.1 Participant Overview

Table 7.1 summarizes the participants and their assigned conditions. Wizards are identified by code to protect confidentiality. All six participants were Bucknell University professors drawn from Computer Science, Chemical Engineering, Digital Humanities, and Logic and Philosophy of Science. Demographic information (programming background) was collected during recruitment.

ID	Condition	Background	Programming Experience
W-01	Choregraphe	Digital Humanities	None
W-02	HRIStudio	Logic and Philosophy of Science	Moderate
W-03	Choregraphe	Computer Science	Extensive
W-04	Choregraphe	Chemical Engineering	Moderate
W-05	HRIStudio	Chemical Engineering	None
W-06	HRIStudio	Computer Science	Extensive

Table 7.1: Summary of wizard participants and assigned conditions.

Table 7.2 presents the primary outcome scores, which are discussed next.

7.2 Primary Measures

ID	Condition	DFS	ERS	SUS
W-01	Choregraphe	42.5	65	60
W-02	HRIStudio	100	95	90
W-03	Choregraphe	65	60	75
W-04	Choregraphe	62.5	75	42.5
W-05	HRIStudio	100	95	70
W-06	HRIStudio	100	100	70

Table 7.2: Primary outcome scores by wizard and condition.

7.2.1 Design Fidelity Score (DFS)

The Design Fidelity Score measures how completely and correctly each wizard implemented the written specification of their assigned experiment. Scores range from 0 to 100, with full points awarded only when a component — a rubric criterion representing a required speech action, gesture, or control-flow element — is both present and correct. (For a full description of rubric categories, see Section 6.7.)

Across the six participants, DFS scores divided sharply by study condition: all

three HRIStudio wizards achieved a perfect score of 100, while the three Choregraphe wizards scored 42.5, 65, and 62.5. The following paragraphs describe the key findings from each session.

W-01 received a DFS of 42.5. Analysis of the exported project file found all four interaction steps present and correctly sequenced; the conditional branch was wired and functional. Speech fidelity was partial: W-01 deviated from the specification by substituting a different rock color in the narrative and comprehension question, departing from the "red" specified in the paper protocol. Items 1 and 4 (introduction and branch responses) received full points; items 2 and 3 received half points due to the content mismatch. The gesture category scored zero. Both the introduction wave and the narrative gesture were implemented via the tool's *Animated Say* function, which generates motion non-deterministically from a library rather than placing a specific gesture node; under the rubric's clarifying rule, this does not satisfy the Correct criterion. Item 7 (nod or head shake) was not explicitly programmed. The control-flow category was split: item 9 (correct step sequence) received full points; item 8 (conditional branch) received half points because the branch was resolved by manually deleting and re-routing connections during the trial rather than through a dedicated conditional node wired at design time.

W-02 received a DFS of 100. The exported project file confirmed all four interaction steps present and correctly sequenced, speech content matching the written specification verbatim, gestures placed using dedicated action nodes, and the conditional branch wired through HRIStudio's branch component. No tool-operation interventions were logged during the design phase. W-02 completed the design in 24 minutes, within the 30-minute allocation.

W-03 received a DFS of 65. W-03 approached the design as a block programming exercise, constructing extra nodes and attempting a concurrent execution structure not called for by the specification. One C-type clarification (see Section 6.7) was required: I noted that control-flow logic relying on onboard speech recognition was outside the scope of this study, since Wizard-of-Oz execution routes all speech decisions through the wizard rather than the robot. Speech fidelity was partial: two of the three scorable speech items were present, with not all delivered correctly. No conditional branch was implemented in the final design, resulting in zero points for that category. The design phase extended to 37 minutes, seven minutes over the 30-minute allocation.

W-04 received a DFS of 62.5. The design phase ran 35 minutes without reaching completion, making W-04 the only wizard in the study who did not finish the design before the cutoff. Four T-type tool-operation interventions and one C-type clarification were logged. During training, W-04 asked about running two behavior blocks simultaneously and how to edit a block, reflecting early engagement with Choregraphe's concurrent flow model. During the design phase, W-04 asked about interpretation of punctuation in speech content, generating three simultaneous T-type marks across items 1–3. W-04 also independently attempted to use Choregraphe's choice block for conditional branching; the block did not execute correctly. The researcher re-explained the WoZ execution model and how to branch by manual step selection. Speech items 1, 2, and 4 received full points; item 3 (the comprehension question) was absent from the final design. Gesture items 5 and 6 received full points; item 7 (nod or head shake) was present but not marked correct (5/10). The conditional branch received zero points; no functional branch was wired at export. Step sequencing received partial credit (7.5/15).

W-05 received a DFS of 100. The design phase completed in 18 minutes, the shortest design phase in the study. Training concluded in 6 minutes with no questions asked; the wizard described the platform as "pretty straightforward." Two T-type interventions and three C-type clarifications were logged during the design phase. The T-type interventions concerned editing properties in the right pane of the experiment designer and understanding that the branch block requires predefined steps; both were addressed without affecting the final design. The C-type clarifications concerned what "steps" represent as structural containers, the relationship between the written specification's speech and platform speech actions, and a related conceptual question. The wizard added a creative narrative gesture not specified in the protocol (a crouch animation); this was present and correct under the rubric. The DFS assessment noted that the wizard's design mapped well from the specification.

W-06 received a DFS of 100. Two T-type interventions were logged during the design phase, both pertaining to item 6 (narrative gesture): at 15:21, W-06 attempted to use parallel execution for a gesture action and was unable to edit the action node; at 15:24, W-06 encountered difficulty resetting the robot's posture and was directed to recommended posture blocks. In both cases, W-06 resolved the issue independently after the initial prompt. W-06's programming background led to a more elaborate design than the specification required, including extra posture-reset actions that were ultimately redundant since the robot was already in the correct starting position; these additions did not affect scoring since all required actions were present and correct in the exported project file. The conditional branch was wired correctly, and all speech and gesture items matched the specification. W-06 completed the design in 21 minutes, within the 30-minute allocation.

Across the three HRIStudio sessions, DFS scores were 100, 100, and 100 (mean 100). Across the three Choregraphe sessions, DFS scores were 42.5, 65, and 62.5 (mean 56.7).

7.2.2 Execution Reliability Score (ERS)

The Execution Reliability Score measures how faithfully the designed interaction executed during the live trial.

Execution results followed the same pattern as design fidelity. HRIStudio trials produced ERS scores of 95, 95, and 100, with no session requiring tool-operation guidance to reach the interaction's conclusion. Choregraphe trials averaged 66.7, with branching failures in two of three sessions and a speech content deviation in the third (see Section 7.4 for details). The per-session details are as follows.

W-01 received an ERS of 65. The trial ran for approximately five minutes. In this session, I served as the test subject during the live trial. Through that experience I confirmed that a separately recruited participant is not required: the DFS and ERS both evaluate the wizard's implementation and execution fidelity rather than a subject's behavioral responses. Subsequent sessions therefore ran the trial phase with the wizard executing the designed interaction directly, without a separate test subject. The introduction speech and gesture executed correctly. The narrative speech executed but deviated from the specification due to the modified rock color described above. The comprehension question was delivered, a branch response was triggered, and the interaction proceeded to its conclusion. Gesture synchronization was partial: a pause gesture executed, but coordination between speech and movement was

inconsistent at several points. No system disconnections or crashes occurred.

W-02 received an ERS of 95. The trial ran for approximately five minutes. Introduction speech and gesture, narrative speech, comprehension question, and branch response content all executed correctly and matched the specification. During the trial, the interaction briefly advanced to an incorrect step when a branch transition misfired; this was immediately corrected by manually selecting the correct step in the execution interface. This incident was logged as an H-type intervention (platform behavior, not wizard error). The branching item scored 5 out of 10 on its own merits: the branch was present in the design and execution reached the branch step, but the initial misfire meant the transition was not fully correct before manual correction. No other deviations or system failures occurred.

W-03 received an ERS of 60. The trial ran for approximately five minutes. Speech execution was partial: two of three items were present but not all delivered correctly. Gesture and speech synchronization was poor throughout the interaction; motion cues were present but did not coordinate reliably with corresponding speech actions. The conditional branch, absent from W-03's design, was not executed during the trial; the interaction proceeded without a branch resolution step. No system disconnections or crashes occurred.

W-04 received an ERS of 75. The trial ran for approximately four minutes. Introduction and narrative speech executed correctly. The comprehension question, absent from the design, was not delivered; the interaction proceeded directly to the branch step. A T-type trial intervention was required to remind W-04 how to trigger the branch; the yes-branch response was delivered following that prompt, capping item 4 at 5/10 (T-assisted). Gesture execution was strong: introduction wave, narrative

gesture, and nod or head shake all executed correctly. Speech and gesture synchronization scored full points. The pause before the comprehension question scored zero, as no question was delivered. No system errors occurred.

W-05 received an ERS of 95. The trial ran for approximately four minutes and reached step 4. The researcher's answer was "Red" (the correct answer), and branch A fired via programmed conditional logic. All speech items executed correctly. Introduction gesture, nod or head shake, speech synchronization, and the pre-question pause all scored full points. One trial intervention pair was logged: the researcher briefly forgot they were in live execution (G-type), then was reminded and manually skipped a non-functional crouch action (T-type, capping item 6 at 5/10). The crouch animation exists in HRIStudio's action library but does not execute on the NAO6 robot-side; skipping it was the correct recovery. All other items scored full points and no system errors occurred. The overall ERS assessment recorded that the interaction executed as designed.

W-06 received a perfect ERS of 100. The trial ran for approximately three minutes. No interventions of any type were logged during the trial phase. All speech items executed correctly and matched the specification. Gestures, speech synchronization, and the pre-question pause all scored full points. The conditional branch was present in the design and fired correctly during execution via programmed conditional logic. The interaction reached its conclusion without errors, disconnections, or researcher involvement.

Across the three HRIStudio sessions, ERS scores were 95, 95, and 100 (mean 96.7). Across the three Choregraphe sessions, ERS scores were 65, 60, and 75 (mean 66.7).

7.2.3 System Usability Scale

The System Usability Scale (SUS) uses a 0–100 scale with a conventional average of 68; scores above 68 indicate above-average perceived usability [29]. W-01 rated Choregraphe with a SUS score of 60. A score of 60 suggests that W-01, a Digital Humanities faculty member with no programming background, found Choregraphe marginal in usability; this outcome is consistent with the high volume of interface-level help requests observed during the design phase.

W-02 rated HRIStudio with a SUS score of 90, the highest score in the study. W-02, a Logic and Philosophy of Science faculty member with moderate programming experience, completed the design phase without tool-operation assistance and rated the platform favorably across usability dimensions.

W-03 rated Choregraphe with a SUS score of 75. W-03, a programmer with prior experience in block programming environments, perceived the tool positively in general terms, framing it as a capable system for its category. Post-session comments indicated that W-03 found the tool harder to apply to this specific task than its general capability suggested, particularly given the WoZ framing's constraint against onboard control-flow logic. W-03 had no prior knowledge of HRIStudio, providing no comparative baseline for their usability rating.

W-04 rated Choregraphe with a SUS score of 42.5, the lowest score in the study. Researcher notes recorded that W-04 attempted the task with evident self-driven engagement but that the platform appeared to get in the way. The gap between effort and outcome in W-04's session, a motivated wizard who exceeded the time allocation without completing the design and required four T-type interventions, is

directly reflected in this rating.

W-05 rated HRIStudio with a SUS score of 70. Post-session comments recorded no issues. W-05, a Chemical Engineering faculty member with no programming background, completed the design well within the allocation and ran the trial to its conclusion without tool-operation difficulty during execution.

W-06 rated HRIStudio with a SUS score of 70. W-06, a Computer Science faculty member with extensive programming experience, completed the design within the allocation and ran a perfect trial without researcher intervention. The score matches W-05's rating exactly; both wizards found the platform above-average in usability despite approaching the task from very different programming backgrounds.

HRIStudio study condition SUS scores were 90, 70, and 70 (mean 76.7). Choregraphe study condition SUS scores were 60, 75, and 42.5 (mean 59.2).

Figure 7.1 summarizes the three primary measures side-by-side. In each group, the left bar represents the Choregraphe mean and the right bar represents the HRIStudio mean. HRIStudio exceeds Choregraphe on every measure, with the largest gap on DFS (43.3 points) and the smallest on SUS (17.5 points).

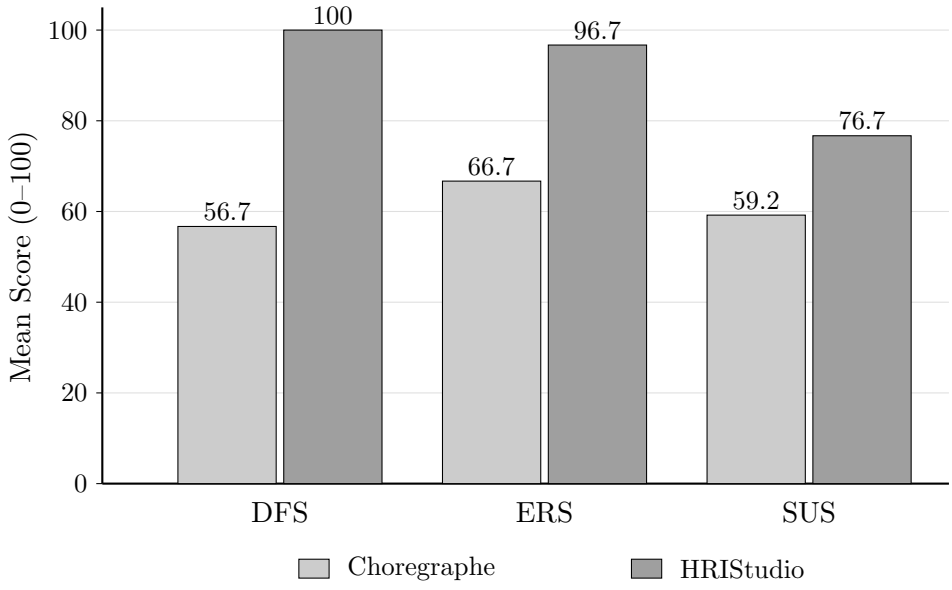


Figure 7.1: Mean scores by condition across the three primary outcome measures.

7.3 Supplementary Measures

7.3.1 Session Timing

W-01's design phase extended to 35 minutes, five minutes over the 30-minute allocation, compressing the trial and debrief to 5 minutes each. Despite this, W-01 declared the design complete rather than abandoning it, and the robot executed a recognizable version of the specification during the trial.

W-02's training phase concluded in 7 minutes, roughly half the standard 15-minute allocation. This reflects HRIStudio's more intuitive onboarding rather than simply W-02's technical background: the platform's guided workflow and timeline-based model required less explanation before the wizard was ready to begin the design phase. W-02's design phase then concluded in 24 minutes, within the allocation, and the trial ran for approximately five minutes.

W-03's design phase extended to 37 minutes, the longest design phase in the study, despite W-03's programming background. The overrun reflects not conventional interface friction but the time spent constructing and then revising an over-engineered design; beginning sessions from W-02 onward enforced the 30-minute transition, so W-03's overrun constitutes a procedural exception noted in the observer log.

W-04's design phase ran 35 minutes without completion, the only session in which the wizard did not finish before the cutoff. Training took 17 minutes, the longest training phase in the study; W-04 entered the design phase with questions about concurrent block execution that presaged later difficulties with branching.

W-05's design phase completed in 18 minutes, the shortest in the study. The overall session lasted 32 minutes, also the shortest. Training took 6 minutes with no questions asked. The contrast between W-04 and W-05 is striking: both come from Chemical Engineering, both with no robotics background, yet the difference in assigned tool produced a 17-minute gap in design completion time and a qualitatively different session experience.

W-06's training phase concluded in 8 minutes and the design phase completed in 21 minutes, both within their allocations. The overall session lasted 37 minutes. The trial ran for approximately three minutes, the shortest trial phase in the study, reflecting a clean execution without errors or researcher interventions.

Across all six sessions, Choregraphe design phases averaged approximately 35.7 minutes; W-01 and W-03 exceeded the 30-minute target but completed their designs before the session time limit, while W-04 was the only wizard cut off by the limit without finishing. HRIStudio design phases averaged 21 minutes across three sessions,

all within the allocation. Training phases similarly diverged: Choregraphe training averaged approximately 14.7 minutes, while HRIStudio training averaged 7 minutes.

Figure 7.2 compares the per-condition means for training, design, and total session duration. The gap is concentrated in the design phase and carries through to the total session length; training duration also diverges, with Choregraphe wizards requiring roughly twice as long to reach readiness.

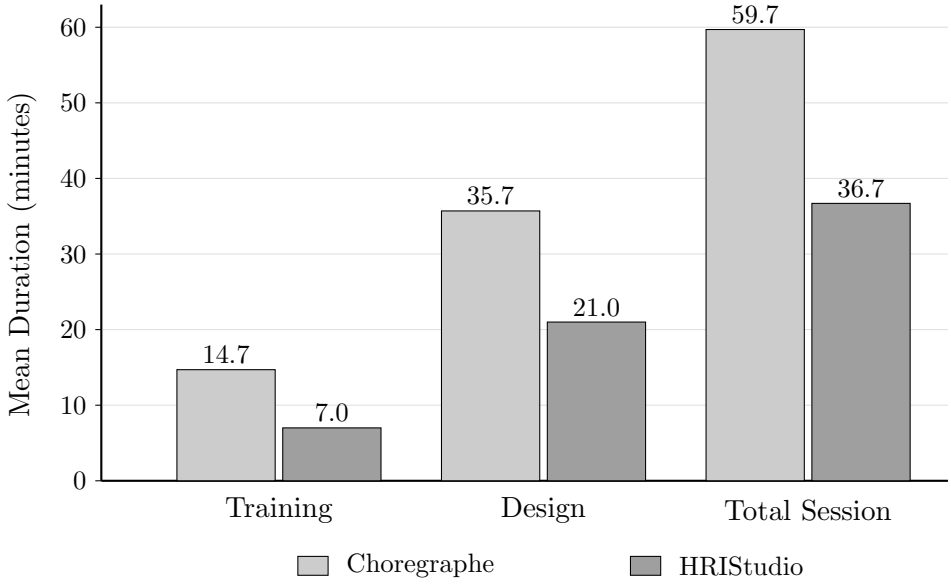


Figure 7.2: Mean phase durations by condition.

7.3.2 Intervention Log

W-01 generated a high volume of help requests during the design phase, primarily concerning Choregraphe's interface rather than the specification itself. The wizard demonstrated understanding of the task but encountered repeated friction with the tool's connection model, behavior box configuration, and branch routing. This pattern, understanding the goal but struggling with the mechanism, is characteristic of

the accessibility problem described in Chapter 2.

W-02 generated minimal interventions. No T-type tool-operation assistance was required during the design phase; the wizard navigated HRIStudio's interface without guidance. One H-type intervention was logged during the trial phase, corresponding to the branch step misfire described in the ERS section above.

W-03 generated one C-type intervention during the design phase: a clarification that control-flow logic dependent on onboard speech recognition was outside the study's scope. No T-type interventions were required; W-03 navigated Choregraphe independently throughout the design phase. The absence of T-type interventions for W-03, compared to W-01's high T-type volume, suggests that programming background moderates the interface accessibility problem in Choregraphe: the tool does not block programmers the way it blocked a non-programmer, though it still produced a lower DFS than HRIStudio.

W-04 generated the highest T-type count in the Choregraphe study condition: five total design-phase interventions (4 T-type, 1 C-type), plus one T-type intervention during the trial. The design-phase T marks covered speech content punctuation ($\times 3$, items 1–3) and the failed choice block attempt (item 8). The pattern echoes W-01's volume of tool-level friction, concentrated in a wizard with moderate rather than no programming experience.

W-05 generated five design-phase interventions (2 T-type, 3 C-type) and two trial interventions (1 T-type, 1 G-type). The design-phase T marks concerned interface orientation (right-pane editing, branch block configuration); the C-type clarifications concerned conceptual mappings between the written specification and HRIStudio's

structural model. Importantly, none of the clarifications blocked design completion, and the final DFS was unaffected. The C-type pattern for W-05 reflects a different kind of engagement from Choregraphe's T-type pattern: questions about what the tool means rather than how to operate it.

W-06 generated two T-type interventions during the design phase, both pertaining to item 6 (narrative gesture): one for an attempted use of parallel action execution, and one for difficulty resetting the robot's posture, for which specific recommended blocks were suggested. W-06 resolved both issues independently after the initial prompts. No interventions of any type were logged during the trial phase, making W-06 the only wizard in the study to complete the trial with zero interventions.

7.4 Qualitative Findings

7.4.1 Observed Specification Deviation

A notable qualitative finding from W-01's session was an unprompted deviation from the written specification: the wizard substituted a different rock color in the robot's speech and comprehension question, departing from the "red" specified in the paper protocol. This was not a tool failure; the wizard made a deliberate creative choice that the tool did not prevent or flag. The deviation was undetected until the live trial, when it surfaced during execution. This incident illustrates the reproducibility problem concretely: without automated protocol enforcement, wizard behavior can drift from the specification in ways that are invisible until execution, affecting the validity of the resulting interaction data.

No specification deviations from the written protocol were observed in W-02, W-04, W-05, or W-06. W-03 introduced extra nodes beyond the specification's scope, which was addressed by a C-type clarification during design. W-05 added a creative gesture not required by the specification (crouch), which was not a deviation from the protocol's content but an elaboration of the gesture category; it scored within the rubric and was noted for completeness. The speech substitution incident in W-01 remains the only case of content drift from the written specification, and it occurred exclusively in the Choregraphe study condition.

7.4.2 Wizard Experience

W-01 expressed that the training was comprehensible and that the underlying logic of the task was clear. The primary source of frustration was Choregraphe's interface for handling conditional branches and managing the timing of parallel behaviors. Post-session comments suggested that the wizard would not use Choregraphe independently for future HRI work without technical support.

W-02 engaged with HRIStudio's timeline-based interface without requiring tool-operation guidance. The session proceeded efficiently, and W-02's Logic and Philosophy of Science background, combined with moderate programming experience, appeared to support both the technical implementation and the contextual understanding of the interaction scenario. No notable sources of friction were observed during design or trial phases.

W-03 approached the task as a programming challenge, applying Choregraphe's full feature set beyond what the specification required. When the WoZ framing was

clarified (specifically that branching should reflect wizard decisions rather than on-board robot logic), W-03 revised the design but the over-engineered structure introduced earlier persisted in the final export and was reflected in the DFS score. W-03 described Choregraphe as a powerful block programming environment, but noted that applying it to this task was harder than its general capability implied, a characterization consistent with the tool-task mismatch the study is designed to surface.

W-04 approached the session with clear engagement and self-driven exploration: independently attempting Choregraphe features (concurrent blocks, choice node) that went beyond what prior instructions had covered. The researcher noted "Great attempt. Self-driven to explore." The SUS score of 42.5 reflects a session where ambition consistently exceeded what the tool's interface could support without additional guidance. W-04's post-session comment that quality was attempted but the platform got in the way is arguably the most direct characterization of the accessibility problem in the dataset.

W-05 presented the clearest demonstration of HRIStudio's accessibility case. With no programming background, W-05 trained in 6 minutes, asked no questions, completed the design in 18 minutes with a creative addition, and ran the trial to completion. The researcher's session notes observed: "Overall good session. Learning: different backgrounds determine tool curiosity and drive to self-explore." W-05's willingness to add a crouch gesture beyond the specification, and their straightforward navigation of the platform without tool-operation confusion, suggests that HRIStudio's design model successfully supports exploratory use by non-programmers without producing the friction pattern observed in the Choregraphe study condition.

W-06 approached the design with a programmer's instinct for thoroughness, ini-

tially exploring parallel execution structures for gesture actions and adding posture-reset steps beyond what the specification called for. The two T-type design-phase interventions reflected this exploratory behavior rather than confusion about the task. The extra posture-reset actions in the final design were redundant in practice since the robot was already in the correct starting position, but they did not interfere with the required items and the design achieved a perfect DFS. W-06's trial ran entirely without researcher intervention, producing the only perfect ERS in the study. The session illustrates a different accessibility profile from W-05: where W-05 encountered no interface friction at all, W-06's programming background produced brief exploratory detours that the platform absorbed without compromising the final design or execution.

7.5 Chapter Summary

Across all six sessions, the evidence consistently favored HRIStudio on every primary and supplementary measure. Every HRIStudio wizard produced a perfect design without tool-operation assistance, while all three Choregraphe wizards scored below perfect and the only wizard who did not finish before the session cutoff was in the Choregraphe study condition. On execution consistency, HRIStudio trials reached their conclusion without researcher guidance in every case; Choregraphe produced branching failures in two of three sessions and a content deviation in the third (see Section 7.4). Perceived usability followed the same split, with all HRIStudio ratings above the SUS average and all Choregraphe ratings below it. Taken together, these results suggest that HRIStudio's design principles produce measurable gains in both accessibility and execution consistency compared to standard practice. Chapter 8

interprets these findings in the context of the research questions.

Chapter 8

Discussion

This chapter interprets the results presented in Chapter 7 against the two research questions established in Chapter 6, situates the findings within the broader literature on WoZ methodology, and identifies the limitations of this study.

8.1 Interpretation of Findings

8.1.1 Research Question 1: Accessibility

The first research question asked whether HRIStudio enables domain experts without prior robotics experience to successfully implement a robot interaction from a written specification. The Choregraphe study condition provides the baseline against which this question is evaluated.

The six completed sessions provide directional evidence on the accessibility question. Across the three Choregraphe wizards, design fidelity scores were 42.5, 65, and 62.5, yielding a Choregraphe mean of 56.7. Across the three HRIStudio sessions, all three wizards achieved a DFS of 100. No HRIStudio wizard required a T-type intervention that reflected an inability to operate the platform; the T-type marks logged for W-05 concerned interface orientation, and those logged for W-06 concerned gesture execution details (parallel execution and posture-reset blocks), neither of which constituted fundamental operational barriers. By contrast, Choregraphe produced design difficulties across all three sessions. W-01 required T-type assistance for connection routing and branch wiring. W-03 required no T-type interventions but over-engineered the design, adding concurrent execution nodes and attempting onboard speech-recognition logic that falls outside the WoZ paradigm. W-04 required T-type assistance for speech content punctuation and a failed choice block attempt.

The SUS scores reinforce this pattern. Choregraphe SUS scores were 60, 75, and 42.5 (mean 59.2), all at or below the average usability benchmark of 68 [29]. HRIStudio SUS scores were 90, 70, and 70 (mean 76.7), all above the benchmark. The Choregraphe study condition produced the lowest single SUS score in the study (42.5, W-04), a wizard who described the platform as getting in the way of their attempt. The HRIStudio study condition produced the highest (90, W-02). Because assignment was stratified by programming background, each condition contains exactly one wizard with *None* experience, one with *Moderate* experience, and one with *Extensive* experience, enabling a direct cross-background comparison: W-01 (*None*, Choregraphe, SUS 60) versus W-05 (*None*, HRIStudio, SUS 70); W-04 (*Moderate*, Choregraphe, SUS 42.5) versus W-02 (*Moderate*, HRIStudio, SUS 90); W-03 (*Extensive*, Choregraphe, SUS 75) versus W-06 (*Extensive*, HRIStudio, SUS 70). HRIStudio

scores exceed Choregraphe scores at the *None* and *Moderate* levels; at the *Extensive* level the scores reverse by five points, suggesting that extensive programming experience largely attenuates the tool-level usability difference. It is worth noting that only one participant (W-01, Digital Humanities) came from a non-STEM discipline; the remaining five wizards held backgrounds in Computer Science, Chemical Engineering, or Logic and Philosophy of Science, a composition that limits claims about accessibility for humanities-domain researchers.

The most striking accessibility finding comes from W-05: a Chemical Engineering faculty member with no programming experience trained in 6 minutes, completed a perfect design in 18 minutes with no operational confusion, and ran the trial to conclusion. This outcome directly addresses the accessibility research question. HRIStudio's timeline-based model and guided workflow allowed a domain novice to implement the written specification correctly on their first attempt, without the interface friction that blocked or slowed all three Choregraphe wizards. Session timing data underscores the difference: Choregraphe design phases averaged 35.7 minutes (two overruns, one incomplete), while HRIStudio design phases averaged 21 minutes (all three within the allocation). Underlying this difference is a structural property of the two tools: HRIStudio's model is domain-specific to Wizard-of-Oz execution, so wizard effort is channeled toward implementing the specification more completely rather than elaborating the tool's architecture. Choregraphe's general-purpose programming model makes the opposite available, and both W-03 and W-04 took it, spending time on concurrent execution structures and a speech-recognition-driven choice block that the WoZ context does not support. No HRIStudio wizard had that option, and all three scored 100 on the DFS.

8.1.2 Research Question 2: Reproducibility

The second research question asked whether HRIStudio produces more reliable execution of a designed interaction compared to Choregraphe. The most instructive finding from W-01's session is not a score but an incident: the wizard deviated from the written specification by substituting different speech content, and this was not flagged or caught until the live trial (see Section 7.4 for the full account).

This is precisely the failure mode the reproducibility problem predicts. Riek's [14] review found that fewer than 4% of published WoZ studies reported any measure of wizard error, meaning most studies have no mechanism to detect whether execution matched design intent. W-01's session demonstrates that such deviations occur even in controlled conditions with a single, simple specification and an engaged wizard. The deviation was not negligence; it was creative drift made possible by a tool that places no structural constraint on what the wizard types into a speech action.

HRIStudio's protocol enforcement model is designed to prevent this class of deviation by locking speech content at design time. The available data supports this design intent. No speech content deviations occurred in any of the three HRIStudio sessions. W-05 added an action beyond the specification (a crouch gesture), but this was an elaboration of the gesture category rather than a substitution of specified content, and it was scored within the rubric. The Choregraphe study condition produced the only speech substitution in the dataset (W-01) and two sessions in which branching was absent from the design entirely (W-03, W-04).

ERS scores reflect the downstream effect of these design differences. Choregraphe ERS scores were 65, 60, and 75 (mean 66.7). HRIStudio ERS scores were 95, 95, and

100 (mean 96.7). The branching item is particularly instructive: in the Choregraphe study condition, branch execution was either absent from the design entirely (W-03) or present but not implemented as conditional logic (W-01, W-04). W-01 resolved the branch by manually re-routing connections during the trial; W-04 required a T-type trial intervention to be reminded how to trigger the branch step. In all three HRIStudio sessions, the conditional branch was present in the design and executed during the trial. W-05's branch fired cleanly via programmed conditional logic; W-02's session saw a brief platform-side step misfire immediately corrected by manual step selection, logged as an H-type (platform behavior) intervention rather than a wizard error; W-06's branch fired cleanly with no intervention of any kind. In no HRIStudio session did branch execution depend on tool-operation guidance from the researcher.

8.1.3 Session Timing and Downstream Effects

W-01's design phase extended to 35 minutes, overrunning the 30-minute allocation by five minutes and leaving approximately five minutes for the trial phase. It is worth distinguishing between the two factors at play here: the overrun reflected both the tool's demands on the wizard and a procedural decision not to interrupt W-01 at the 30-minute mark. Subsequent sessions enforced the transition to the trial phase at 30 minutes regardless of design completion status, consistent with the observer protocol. That said, if a tool's demands make design completion within the allocation genuinely difficult, the risk of an overrun is real regardless of enforcement: a wizard who has not finished at 30 minutes faces a reduced trial window no matter when the cutoff is applied.

Across all six sessions, design phase overruns are concentrated in the Choregraphe study condition. W-01 and W-03 each exceeded the 30-minute design target but completed their designs before the session time limit; W-04 was the only wizard cut off by the limit without finishing. No HRIStudio wizard exceeded the target. This pattern holds across programming backgrounds: W-01 (non-programmer) and W-03 (extensive programmer) both overran in the Choregraphe study condition, while W-05 (non-programmer, HRIStudio) completed in 18 minutes and W-06 (extensive programmer, HRIStudio) completed in 21 minutes. The timing data thus corroborates the DFS and SUS findings as a supplementary accessibility indicator, and supports the conclusion that the overrun pattern is attributable to assigned tool rather than wizard background alone. Because programming experience was balanced across conditions by stratified assignment, the design-phase timing difference cannot be attributed to prior programming experience.

8.2 Comparison to Prior Work

Because assignment was stratified by programming experience, the overall 17.5-point gap in both means reflects a genuine tool-level effect rather than a sampling artifact. Pot et al. [10] introduced Choregraphe as a tool for enabling non-programmers to create NAO behaviors, but subsequent HRI research has treated it primarily as a programmer's tool in practice. This study confirms that characterization: W-01 (no programming experience) and W-04 (moderate experience) both required substantial T-type assistance and produced incomplete or deviation-prone designs, while W-03 (extensive experience) navigated the interface without T-type support yet still over-engineered the design and scored below every HRIStudio participant on both

DFS and ERS. Riek's [14] observation that WoZ tools tend to require substantial technical investment even when the underlying experiment is conceptually simple holds across all three Choregraphe sessions regardless of background. In contrast, the HRIStudio results support the claim advanced in prior work [3, 4] that a domain-specific, web-based platform can decouple task complexity from interface complexity: all three HRIStudio wizards—spanning no, moderate, and extensive programming experience—achieved a perfect DFS, and none encountered a fundamental barrier to operating the platform.

The specification deviation in W-01's session connects directly to Porfirio et al.'s [28] argument that formal, verifiable behavior specifications are a prerequisite for reproducible HRI. Porfirio et al. propose specification languages as the solution; HRIStudio takes a complementary approach by embedding the specification into the execution environment, making deviation structurally harder rather than formally detectable after the fact. The ERS data confirms this design intent: no speech content deviations occurred across all three HRIStudio sessions, and the HRIStudio ERS mean of 96.7 versus 66.7 for Choregraphe supports the conclusion that structural enforcement produces more reliable execution in practice. Riek's [14] finding that only 3.7% of published WoZ studies reported any measure of wizard error makes this comparison particularly significant: the ERS operationalizes exactly the kind of execution measurement the literature has consistently omitted, and the difference it surfaces here is substantial.

The SUS scores are consistent with prior tool evaluations in HCI. The Choregraphe mean of 59.2 falls below the average benchmark of 68 [29] and below scores reported for general-purpose visual programming environments in comparable studies, consistent

with Bartneck et al.'s [1] finding that domain-specific design is necessary to make tools genuinely accessible to non-programmers. The HRIStudio mean of 76.7 places the platform above the benchmark across all three sessions. Because programming experience is balanced across conditions by design, the overall 17.5-point gap in the two conditions' means reflects a genuine tool-level effect rather than an artifact of the sample's background composition. The gap is largest at the Moderate experience level (W-02 HRIStudio 90 vs. W-04 Choregraphe 42.5) and smallest at the Extensive level, where the scores reverse by five points (W-03 Choregraphe 75 vs. W-06 HRIStudio 70), suggesting that extensive programming experience largely attenuates the tool-level usability difference while the accessibility advantage remains pronounced for non-programmers and moderate programmers.

8.3 Limitations

This study has several limitations that must be considered when interpreting the findings.

Sample size. With six wizard participants ($N = 6$), the study is too small for inferential statistics. The reported scores are descriptive. Patterns in the data can suggest directions for future work but cannot establish causal claims about the effect of the tool on design fidelity or execution reliability.

Trial execution without a separate test subject. Following scheduling difficulties, the study protocol was adjusted so that the wizard executes the designed interaction directly rather than running it for a separate test subject. Because the

DFS and ERS are scored against the exported project file and live observation rather than a subject's behavioral responses, this change does not affect the primary quantitative measures. The trial phase evaluates whether the wizard's design executes as specified; the presence or absence of a separate subject does not alter that criterion.

Single task. Both study conditions used the same Interactive Storyteller specification. While this controls for task difficulty, it limits generalizability. The task is simple relative to real HRI experiments; the gap between conditions may be larger or smaller with a more complex protocol involving multiple branches or longer interaction sequences.

Uncontrolled dimensions. Programming experience was balanced across conditions by stratified assignment (see Section 6.7 and Chapter 6): each condition contains one wizard at each of the three experience levels (*None*, *Moderate*, *Extensive*). This controls for programming background as a potential confounder but does not extend to other dimensions. The small N means that balance on other potentially relevant dimensions (disciplinary background, prior experience with visual programming tools, or familiarity with robots more broadly) was not assessed or controlled and remains a source of variability not addressed in this pilot.

Platform version. HRIStudio is continuously evolving. The version used in this study represents the system at a specific point in time. Future iterations may change how the wizard interface presents protocol steps, how branch conditions are constructed during the design phase, or how protocol enforcement is applied during execution. Any of these changes could affect how easily a non-programmer completes the design challenge or how reliably the tool enforces the specification during the trial, potentially altering the DFS and ERS scores observed under otherwise identical

conditions. Results from this study therefore describe the system as it existed at the time of data collection and may not generalize to later releases.

8.4 Chapter Summary

This chapter interpreted the results of all six completed pilot sessions against the two research questions and connected the findings to prior work. Across all primary measures, the directional evidence favors HRIStudio. HRIStudio wizards uniformly achieved perfect design fidelity (DFS 100) and near-perfect execution reliability (mean ERS 96.7), while Choregraphe wizards averaged DFS 56.7 and ERS 66.7, with design overruns in all three sessions and no session completing without researcher guidance. The W-01 content deviation (see Section 7.4) illustrates the reproducibility problem concretely; its absence in all three HRIStudio sessions is consistent with the enforcement model's design intent. Programming backgrounds are balanced across study conditions by stratified assignment, strengthening the cross-background comparisons. The limitations of this pilot study, including sample size, task simplicity, and the single-session design, are acknowledged and inform the future directions described in Chapter 9.

Chapter 9

Conclusion and Future Work

This thesis set out to address two persistent problems in Wizard-of-Oz-based social robotics research. The first is the Accessibility Problem: a high technical barrier prevents domain experts who are not programmers from conducting HRI studies independently. The second is the Reproducibility Problem: the fragmented landscape of custom tools makes it difficult to verify or replicate experimental results across studies and labs. This chapter summarizes the contributions of the work, reflects on what the pilot study results suggest, and identifies directions for future investigation.

9.1 Contributions

This thesis makes three contributions to the field of HRI research infrastructure.

A principled architecture for WoZ platforms. The primary contribution is a set of design principles for Wizard-of-Oz infrastructure: a hierarchical specification

model (Study → Experiment → Step → Action), an event-driven execution model that separates protocol design from live trial control, and a plugin architecture that decouples experiment logic from robot-specific implementations. These principles are not specific to any one robot platform or research group; they describe a general approach to building WoZ tools that are simultaneously accessible to non-programmers and reproducible across executions. The principles were derived from a systematic analysis of reproducibility failures in published WoZ literature, grounded in the prior work of Riek [14] and Porfirio et al. [28], and refined through the design and implementation process described in Chapters 4 and 5.

HRIStudio: a complete, operational platform. The second contribution is HRIStudio, an open-source, web-based platform that fully realizes the design principles described above. HRIStudio is distributed under the MIT License and available at a public repository. HRIStudio provides a visual experiment designer, a consolidated wizard execution interface, role-based access control for research teams, and a repository-based plugin system for integrating robot platforms including the NAO6 used in this study. HRIStudio demonstrates that the design principles are not only technically feasible but can be delivered as a complete system that real researchers use without programming expertise, making it both an artifact and an instrument of validation. The platform's architecture is documented in detail in Chapter 5 and the accompanying technical appendix.

Pilot empirical evidence. The third contribution is a pilot between-subjects study comparing HRIStudio against Choregraphe as a representative baseline tool. While the pilot scale precludes inferential claims, the study provides directional evidence on both research questions and produces a concrete demonstration of the

reproducibility problem in a controlled setting: a wizard using Choregraphe deviated from the written specification in a way that was undetected until the live trial. This incident motivates the enforcement model at the core of HRIStudio's design and illustrates why the reproducibility problem is difficult to solve through training or norms alone.

9.2 Reflection on Research Questions

The central question this thesis addressed was: *can the right software architecture make Wizard-of-Oz experiments more accessible to non-programmers and more reproducible across participants?* The evidence from the pilot study suggests the answer is yes, with the qualifications appropriate to a small N directional study.

On accessibility, all three HRIStudio sessions produced a perfect DFS of 100, with design phases averaging 21 minutes, all within the allocation. The most direct demonstration comes from W-05: a Chemical Engineering faculty member with no programming background trained in 6 minutes, completed a perfect design in 18 minutes, and ran the trial to conclusion. SUS scores reflect the same directional split: Choregraphe mean 59.2 (below the average benchmark of 68), HRIStudio mean 76.7 (above it).

On reproducibility, the content deviation observed in W-01's Choregraphe session (see Section 7.4) illustrates the failure mode the reproducibility problem predicts. No equivalent deviation occurred in any HRIStudio session. Branching was present and executed correctly in all three HRIStudio trials; ERS means reflect the outcome: 66.7

for Choregraphe, 96.7 for HRIStudio.

9.3 Future Directions

The work described in this thesis suggests several directions for future investigation.

Larger validation study. The most immediate next step is a full-scale study with sufficient participants to support inferential analysis. A sample of 20 or more wizard participants, balanced across programming backgrounds and conditions, would allow the DFS and ERS comparisons to be evaluated for statistical significance. A larger study would also enable subgroup analysis, for example whether the accessibility benefit of HRIStudio is concentrated among non-programmers or extends equally to programmers.

Evaluations with multiple different tasks. The Interactive Storyteller is a simple single-interaction task with one conditional branch. Real HRI experiments are more complex: they involve multiple conditions, longer interactions, and more elaborate branching logic. Evaluating HRIStudio on richer specifications would test whether the accessibility and reproducibility benefits scale with task complexity, and whether any new limitations emerge at that scale.

Longitudinal use. This study evaluated first-session performance, which captures the initial learning curve but not longer-term practice. A longitudinal study tracking wizard performance across multiple sessions would reveal whether HRIStudio's benefits persist or diminish as wizards become proficient, and whether the tool's structured approach continues to enforce reproducibility over time.

Observer and researcher roles. HRIStudio's role-based architecture includes Observer and Researcher roles that were not formally evaluated in this study. Future work should investigate how these roles support team coordination in multi-experimenter studies, and whether the annotation and logging capabilities they enable produce analysis workflows that are meaningfully more efficient than manual video coding.

Platform expansion. The NAO integration used in this study is one instance of HRIStudio's plugin architecture. Extending the plugin ecosystem to include mobile robots, socially assistive robots, and non-humanoid platforms would broaden the system's applicability and test whether the plugin abstraction is sufficiently general to accommodate the range of robot capabilities used in published HRI research.

Community adoption. The reproducibility problem in WoZ research is ultimately a community problem, not a tool problem. Future work should investigate what it would take for HRIStudio to be adopted as shared infrastructure across multiple labs, including documentation standards, experiment sharing mechanisms, and incentive structures that make reproducibility a norm rather than an exception.

9.4 Closing Remarks

The Wizard-of-Oz technique is one of the most powerful tools available to HRI researchers: it allows the study of interaction designs that do not yet exist as autonomous systems, accelerating the feedback loop between design intuition and empirical evidence. But the technique has been practiced for decades without the in-

frastructure needed to make it systematic. Studies are conducted with custom tools that are not always shared, by wizards whose behavior is never verified against a protocol, producing results that cannot be replicated because the exact conditions that produced them were never precisely recorded.

HRIStudio is an attempt to build that infrastructure. It will not solve the reproducibility problem by itself; that requires community norms, institutional incentives, and continued investment in open, shared tooling. But it demonstrates that the technical barriers are not insurmountable: a web-based platform can make WoZ research accessible to domain experts who are not engineers, and execution enforcement can prevent the kinds of specification drift that silently degrade research quality. That is, at minimum, where the work begins.

References

[1] Christoph Bartneck, Tony Belpaeme, Friederike Eyssel, Takayuki Kanda, Merel Keijsers, and Selma Sabanovic. *Human-Robot Interaction – An Introduction*. Cambridge University Press, Cambridge, 2nd edition, 2024.

[2] L. Frank Baum. *The Wonderful Wizard of Oz*. George M. Hill Company, Chicago, IL, 1900.

[3] Sean O'Connor and L. Felipe Perrone. HRIStudio: A Framework for Wizard-of-Oz Experiments in Human-Robot Interaction Studies (Late Breaking Report). 2024 33rd IEEE International Conference on Robot and Human Interactive Communication (RO-MAN), 2024.

[4] Sean O'Connor and L. Felipe Perrone. A Web-Based Wizard-of-Oz Platform for Collaborative and Reproducible Human-Robot Interaction Research. 2025 34th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN), 2025.

[5] Aaron Steinfeld, Odest Chadwicke Jenkins, and Brian Scassellati. The Oz of Wizard: Simulating the Human for Interaction Research. In *Proceedings of the 4th ACM/IEEE International Conference on Human Robot Interaction*, pages

101–108. Association for Computing Machinery, 2009. ISBN 9781605582934. doi: 10.1145/1514095.1514115.

[6] David V. Lu and William D. Smart. Polonius: A Wizard of Oz Interface for HRI Experiments. *Proceedings of the 6th International Conference on Human-Robot Interaction*, pages 77–78, 2011.

[7] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. ROS: An Open-Source Robot Operating System. In *IEEE International Conference on Robotics and Automation*, 2009. URL `https://api.semanticscholar.org/CorpusID:6324125`.

[8] Guy Hoffman and Cynthia Breazeal. OpenWoZ: A Runtime-Configurable Wizard-of-Oz Framework for Human-Robot Interaction. *Proceedings of the 11th ACM/IEEE International Conference on Human-Robot Interaction*, pages 117–124, 2016.

[9] Frank Rietz and Maren Bennewitz. WoZ4U: An Open-Source Wizard-of-Oz Interface for Human-Robot Interaction. In *Proceedings of the 16th ACM/IEEE International Conference on Human-Robot Interaction*, pages 95–103. IEEE, 2021.

[10] E. Pot, J. Monceaux, R. Gelin, and B. Maisonnier. Choregraphe: a graphical tool for humanoid robot programming. In *Proceedings of the 18th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN 2009)*, pages 46–51, 2009. doi: 10.1109/ROMAN.2009.5326209.

[11] John Sören Pettersson and Malin Wik. The longevity of general purpose Wizard-of-Oz tools. In *Proceedings of the Annual Meeting of the Australian Special Interest Group for Computer Human Interaction*, OzCHI '15, pages 422–426, New York, NY, USA, 2015. Association for Computing Machinery. ISBN

9781450336734. doi: 10.1145/2838739.2838825. URL https://doi.org/10.1145/2838739.2838825.

[12] Guillaume Gibert, Morgan Petit, Frederic Lance, Gregoire Pointeau, and Peter F. Dominey. What Makes Humans So Different? Analysis of Human-Humanoid Robot Interaction with a Super Wizard of Oz Platform. In *Proceedings of the 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 931–938, 2013. doi: 10.1109/IROS.2013.6696465.

[13] Anna Helgert, Christopher Straßmann, and Sabine C. Eimler. Unlocking Potentials of Virtual Reality as a Research Tool in Human-Robot Interaction: A Wizard-of-Oz Approach. In *Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction*, pages 123–132, 2024. doi: 10.1145/3610978.3640741.

[14] Laurel D. Riek. Wizard of Oz studies in HRI: a systematic review and new reporting guidelines. *J. Hum.-Robot Interact.*, 1(1):119–136, July 2012. doi: 10.5898/JHRI.1.1.Riek. URL https://doi.org/10.5898/JHRI.1.1.Riek.

[15] Daniel Strazdas, Jonathan Hintz, Anna Maria Felßberg, and Ayoub Al-Hamadi. Robots and Wizards: An Investigation into Natural Human–Robot Interaction. *IEEE Access*, 8:218808–218821, 2020. doi: 10.1109/ACCESS.2020.3042287.

[16] Gavin Bierman, Martín Abadi, and Mads Torgersen. Understanding typescript. In Richard Jones, editor, *ECOOP 2014 – Object-Oriented Programming*, pages 257–281, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. ISBN 978-3-662-44202-9.

[17] Sean O'Connor. HRIStudio: A Web-Based Wizard-of-Oz Platform for Human-

Robot Interaction Research. GitHub repository, 2026. URL https://github.com/soconnor0919/hristudio.

[18] Sean O'Connor. HRIStudio Robot Plugins Repository. GitHub repository, maintained as a submodule of HRIStudio, 2026. URL https://github.com/soconnor0919/robot-plugins.

[19] Anthropic. Claude 3.5 Sonnet. Large Language Model, 2024. URL https://www.anthropic.com/claude.

[20] Anthropic. Claude Code. Agentic CLI Developer Tool, 2025. URL https://www.anthropic.com/claude-code.

[21] SST. OpenCode. Open-source AI Coding Agent, 2024. URL https://opencode.ai.

[22] Google. Gemini CLI. Agentic CLI Developer Tool, 2024. URL https://github.com/google-gemini/gemini-cli.

[23] Google. Antigravity. Integrated Agentic Development Environment, 2025. URL https://antigravity.google.

[24] Zed Industries. Zed Code Editor. High-performance Code Editor with AI Integration, 2023. URL https://zed.dev.

[25] Sean O'Connor. nao6-hristudio-integration: ROS/NAOqi Bridge for HRIStudio. GitHub repository, 2026. URL https://github.com/soconnor0919/nao6-hristudio-integration.

[26] Sean O'Connor. nao-workspace: A Containerized Choregraphe Development Environment. GitHub repository, 2026. URL https://github.com/soconnor0919/nao-workspace.

[27] Guy Hoffman and Xuan Zhao. A primer for conducting experiments in human–robot interaction. *ACM Transactions on Human-Robot Interaction*, 10(3), 2021. doi: 10.1145/3412374.

[28] David Porfirio, Allison Sauppé, Aws Albarghouthi, and Bilge Mutlu. A Framework for Specifying Human-Robot Interaction. *ACM Transactions on Human-Robot Interaction*, 12(1):1–32, 2023.

[29] John Brooke. SUS: A Quick and Dirty Usability Scale. pages 207–212, 1996. doi: 10.1201/9781498710411-35.

[30] Anysphere. Cursor Code Editor. AI-Native Code Editor, 2023. URL `https://cursor.com`.

Appendix A

Blank Study Templates

This appendix contains the blank versions of all study instruments used in the pilot validation study. These templates were used to produce the completed materials in Appendix B.

A note on the Informed Consent Form (ICF): the ICF was submitted with the original protocol to the Bucknell University Institutional Review Board (Protocol #2526-025) and reflects the study design as initially proposed. The protocol was refined before data collection began; the key differences between the ICF and the executed protocol are as follows. First, phase durations were adjusted: Training was planned at 15 minutes, the Design Challenge at 30 minutes, the Live Trial at 10 minutes, and the Debrief at 5 minutes, rather than the 10/20/15/15-minute allocations stated in the ICF. Second, screen recording during the design phase was not implemented, as the DFS is scored from the exported project file rather than from screen footage. Third, the live trial was conducted with the researcher serving as the test

subject rather than a recruited student volunteer, as discussed in Chapter 6. The ODS, DFS, ERS, and SUS templates reflect the protocol as executed.

Contents of this appendix, in order: ODS, DFS, ERS, SUS, ICF

Observer Data Collection Sheet

HRIStudio Pilot Study – Completed by researcher during the live session

Participant ID: _____ **Date:** _____

Condition: ☐ HRIStudio ☐ Choregraphe

Session Start: _____ **Test Subject ID:** _____ **Room:** _____

Test subject type: ☐ Recruited participant ☐ Researcher (foreknowledge — note in session notes)

Phase 1 – Training 15 min planned

Start: _____ **End:** _____ **Actual:** _____ min **Questions asked:** _____

Training notes:

Phase 2 – Design Challenge 30 min planned

Start: _____ **End:** _____ **Time to completion:** _____ min

Completed? ☐ Yes ☐ No (hit time limit)

Intervention log: *Type: **T** = tool operation **C** = task/spec clarification **H** = hardware/tech **G** = general/forgetfulness. T marks are recorded alongside the DFS; they do not affect the DFS score.*

Time	Type	DFS #	Description

Total interventions: _____ **Tool-type (T):** _____ **Other:** _____

Phase 3 – Live Trial 10 min planned

Start: _____ **End:** _____ **Actual:** _____ min **Reached Step 4?** ☐ Y ☐ N

Subject's answer: _____ Branch triggered: ☐ A ☐ B ☐ None/manual

Researcher interventions during trial: *Same type codes as above. T-type interventions cap the relevant ERS item at 50%.*

Time	Type	ERS #	Description

Total trial interventions: _____ **Tool-type (T):** _____

Other issues during trial:

Phase 4 – Debrief 5 min planned

Start: _____ **End:** _____ **SUS completed?** ☐ Yes ☐ No

Project file exported? ☐ Yes ☐ No

Qualitative comments:

Overall Session Notes

Design Fidelity Score Rubric

HRIStudio Pilot Study – Completed by researcher after reviewing the exported project file

Participant ID: _____ **Date:** _____

Condition: ☐ HRIStudio ☐ Choregraphe

Scoring: Full points if Present *and* Correct. Half points if Present but not Correct. Zero if not Present.

Assisted (T): Mark T if a tool-operation intervention was given for this item. Recorded for reference — does *not* affect the Points total or the DFS.

Gestures (items 5–7): Correct only if gesture is a distinct action or animation node separate from the speech action. Auto-motion from "Animated Say" does *not* qualify — it selects movements non-deterministically, so output varies between runs even with the same program.

Branching (item 8): Correct only if a dedicated conditional control-flow node (Switch, Choice, or equivalent) is wired to a voice or input trigger. Manual re-routing during execution does *not* satisfy this criterion.

Component	Present?	Correct?	Assisted?	Points
Speech Actions 40 pts total				
1. Introduction: *"Hello. I want to tell you about Kai..."*	Y / N	Y / N	T / –	/10
2. Narrative: *"Kai had been on the Martian surface for six days..."*	Y / N	Y / N	T / –	/10
3. Question: *"What color was the rock Kai found?"*	Y / N	Y / N	T / –	/10
4. Both branch responses (correct *and* incorrect)	Y / N	Y / N	T / –	/10
Gestures & Actions 30 pts total				
5. Open-hand wave during introduction	Y / N	Y / N	T / –	/10
6. At least one narrative gesture (pause or look down)	Y / N	Y / N	T / –	/10
7. Nod (correct branch) or head shake (incorrect branch)	Y / N	Y / N	T / –	/10
Control Flow & Logic 30 pts total				
8. Conditional branch triggers on participant's answer	Y / N	Y / N	T / –	/15
9. All four steps present in correct sequence	Y / N	Y / N	T / –	/15

Raw Total: _____ / 100 **Tool-assisted items (T, for reference):** _____

Design Fidelity Score (DFS): _____% *DFS = Raw Total. T marks are recorded only; they do not affect the score.*

Notes on Implementation Quality:

Overall Assessment — Did the wizard's design faithfully represent the specification?

Execution Reliability Score Rubric

HRIStudio Pilot Study – Completed by researcher after reviewing the video recording

Participant ID: _____ **Date:** _____

Condition: ☐ HRIStudio ☐ Choregraphe

Trial duration: _____ min **Reached Step 4?** ☐ Y ☐ N **Subject's answer:** _____

Test subject: ☐ Recruited participant ☐ Researcher (note foreknowledge in session notes if Researcher)

Scoring: Full points if Executed *and* Correct *and* not Assisted. Half points if Executed but not Correct, *or* if Assisted (T). Zero if not Executed.

Assisted (T): Mark T only for *tool-operation* interventions during the live trial tied to this item. Design-phase T marks do *not* carry over from the DFS rubric. Do not mark T for hardware failures or general confusion unrelated to tool operation.

Gestures (items 5–7): Correct only if the gesture executed as a distinct, intentional robot movement. Incidental auto-motion during speech does *not* qualify — Animated Say selects movements non-deterministically, so output varies between runs even with an identical program.

Branching (item 4): Correct only if the branch resolved via programmed conditional logic. Manual re-routing or path selection by the wizard during the trial = Executed Y, Correct N.

Behavior	Executed?	Correctly?	Assisted?	Points
Speech Execution 40 pts total				
1. Introduction speech delivered without errors	Y / N	Y / N	T / –	/10
2. Narrative speech delivered without errors	Y / N	Y / N	T / –	/10
3. Comprehension question delivered correctly	Y / N	Y / N	T / –	/10
4. Appropriate branch response given	Y / N	Y / N	T / –	/10
Gesture & Movement Execution 30 pts total				
5. Introduction gesture (open-hand wave) executed	Y / N	Y / N	T / –	/10
6. At least one narrative gesture executed	Y / N	Y / N	T / –	/10
7. Nod or head shake executed correctly	Y / N	Y / N	T / –	/10
Timing & Synchronization 20 pts total				
8. Speech and gestures synchronized	Y / N	Y / N	T / –	/10
9. Pause held before comprehension question	Y / N	Y / N	T / –	/10
System Reliability 10 pts – deduct if errors occur				
10. No disconnections, crashes, or hangs	Y / N	N/A	–	/10

Raw Total: _____ / 100 **Trial-phase assisted (T count):** _____

Execution Reliability Score (ERS): _____% *ERS = Raw Total. Assisted cap applied per-item above.*

Trial-Phase Intervention Log *Tool-type (T) marks reduce the relevant item's score above.*

Time	Type	ERS #	Description

Total trial interventions: _____ **Tool-type (T):** _____

Observed Errors or Issues:

Overall Assessment — Did the interaction execute as the wizard designed it?

System Usability Scale (SUS)

HRIStudio Pilot Study – Completed by wizard after the live trial

Participant ID: _____ **Condition:** ☐ HRIStudio ☐ Choregraphe **Date:** _____

For each statement below, circle the number that best reflects your agreement.
1 = Strongly Disagree 2 = Disagree 3 = Neutral 4 = Agree 5 = Strongly Agree

Statement	1	2	3	4	5
1. I think that I would like to use this system frequently.	○	○	○	○	○
2. I found the system unnecessarily complex.	○	○	○	○	○
3. I thought the system was easy to use.	○	○	○	○	○
4. I think I would need support from a technical person to use this system.	○	○	○	○	○
5. I found the various functions in this system were well integrated.	○	○	○	○	○
6. I thought there was too much inconsistency in this system.	○	○	○	○	○
7. I would imagine that most people would learn to use this system very quickly.	○	○	○	○	○
8. I found the system very cumbersome to use.	○	○	○	○	○
9. I felt very confident using the system.	○	○	○	○	○
10. I needed to learn a lot of things before I could get going with this system.	○	○	○	○	○

Scoring note (for researcher): Odd items are positive; even items are negative. Subtract 1 from each odd item's score; subtract each even item's score from 5. Sum all ten converted scores and multiply by 2.5. Score range: 0–100.

Informed Consent

HRIStudio Pilot Study – Faculty / Wizard Participant

Study. You are invited to participate in a research study conducted by Sean O'Connor (Student PI) and Dr. L. Felipe Perrone (Advisor) in the Department of Computer Science at Bucknell University. The study compares the usability and reproducibility of a new visual tool for robot programming (HRIStudio) against the standard NAO software (Choregraphe).

What you will do. Your session will take approximately **60 minutes**:

1. **Training (10 min)** — Brief tutorial on your assigned software: speech, gestures, and branching logic.
2. **Design Challenge (20 min)** — You will receive a written story specification and implement it as a robot interaction using your assigned tool.
3. **Live Trial (15 min)** — A student volunteer will enter and you will run your program for them.
4. **Debrief (15 min)** — You will complete a short usability survey.

Data collected. Your screen will be recorded during the design phase. The live trial will be video recorded to assess the robot's behavior. All data is stored on encrypted drives; your identity is replaced with a code (e.g., W-01).

Risks and benefits. No risks beyond normal computer use. Your participation contributes to research on accessible tools for human-robot interaction.

Voluntary participation. Participation is entirely voluntary and unrelated to any professional obligations. You may withdraw at any time without penalty.

Questions. Contact Sean O'Connor at `sso005@bucknell.edu` or the Bucknell IRB at `irb@bucknell.edu` (Protocol #2526-025).

_____ _____

Signature Date

_____

Printed Name

Appendix B

Completed Study Materials

This appendix contains the completed study instruments for each of the six sessions conducted prior to the submission of this thesis (W-01 through W-06). The DFS and ERS were scored during and immediately after each session using live observation and the Observer Data Sheet; the SUS was completed by the wizard during the debrief phase.

Contents of this appendix, in order:

- **W-01 (Choregraphe):** ODS, DFS, ERS, SUS
- **W-02 (HRIStudio):** ODS, DFS, ERS, SUS
- **W-03 (Choregraphe):** ODS, DFS, ERS, SUS
- **W-04 (Choregraphe):** ODS, DFS, ERS, SUS
- **W-05 (HRIStudio):** ODS, DFS, ERS, SUS

- **W-06 (HRIStudio):** ODS, DFS, ERS, SUS

Observer Data Collection Sheet

HRIStudio Pilot Study – Completed by researcher during the live session

Participant ID: W01 **Date:** 2026-04-01

Condition: ☐ HRIStudio ☒ Choregraphe

Session Start: 1100 **Test ID:** 001 **Subject Room:** D337

Test subject type: ☐ Recruited participant ☐ Researcher (foreknowledge — note in session notes)

Phase 1 – Training 15 min planned

Start: 1100 **End:** 1115 **Actual:** 15 min **Questions asked:** 4

Training notes: Tooling questions
- Where are block parameters
- Can robot do #s
- how to branch
- how to set up/install tool

Phase 2 – Design Challenge 30 min planned

Start: 1115 **End:** 1150 **Time to completion:** 35 min

Completed? ☒ Yes ☐ No (hit time limit)

Intervention log: *Type: **T** = tool operation **C** = task/spec clarification **H** = hardware/tech **G** = general/forgetfulness. T marks are recorded alongside the DFS; they do not affect the DFS score.*

Time	Type	DFS #	Description
03	T	1	Needed guidance on how to specify text
07	T	5	Re-explanation of animated say
08	T	2	Explanation of say text vs say
11	C	5	Explanation of gesture definition
12	T	4	Re-explanation of branching
15	T	6	Re-direction of motion block location
21	T	8	Re-explanation of block wiring
27	T	9	Explanation of block reordering

Total interventions: 7 **Tool-type (T):** 7 **Other:** 0

Phase 3 – Live Trial 10 min planned

Start: 1150 **End:** 1155 **Actual:** 5 min **Reached Step 4?** ☒ Y ☐ N

Subject's answer: Blue Branch triggered: ☐ A ☐ B ☒ None/manual

Researcher interventions during trial: *Same type codes as above. T-type interventions cap the relevant ERS item at 50%.*

Time	Type	ERS #	Description
3	T	4	When branching, researcher was asked how to jump branches manually.

Total trial interventions: 1 **Tool-type (T):** 1

Other issues during trial: N/A

Phase 4 – Debrief 5 min planned

Start: 1155 **End:** 1200 **SUS completed?** ☒ Yes ☐ No

Project file exported? ☒ Yes ☐ No

Qualitative comments:

Overall Session Notes

Design Fidelity Score Rubric

HRIStudio Pilot Study – Completed by researcher after reviewing the exported project file

Participant ID: W-01 **Date:** 2026-04-01

Condition: ☐ HRIStudio ☒ Choregraphe

Scoring: Full points if Present *and* Correct. Half points if Present but not Correct. Zero if not Present.

Assisted (T): Mark T if a tool-operation intervention was given for this item. Recorded for reference — does *not* affect the Points total or the DFS.

Gestures (items 5–7): Correct only if gesture is a distinct action or animation node separate from the speech action. Auto-motion from "Animated Say" does *not* qualify — it selects movements non-deterministically, so output varies between runs even with the same program.

Branching (item 8): Correct only if a dedicated conditional control-flow node (Switch, Choice, or equivalent) is wired to a voice or input trigger. Manual re-routing during execution does *not* satisfy this criterion.

Component	Present?	Correct?	Assisted?	Points
Speech Actions 40 pts total				
1. Introduction: *"Hello. I want to tell you about Kai..."*	(Y) / N	(Y) / N	(T) / –	5/10
2. Narrative: *"Kai had been on the Martian surface for six days..."*	(Y) / N	(Y) / N	(T) / –	5/10
3. Question: *"What color was the rock Kai found?"*	(Y) / N	(Y) / N	T / –	10/10
4. Both branch responses (correct *and* incorrect)	(Y) / N	(Y) / N	(T) / –	5/10
Gestures & Actions 30 pts total				
5. Open-hand wave during introduction	(Y) / N	Y / (N)	(T) / –	5/10
6. At least one narrative gesture (pause or look down)	(Y) / N	Y / (N)	(T) / –	5/10
7. Nod (correct branch) or head shake (incorrect branch)	Y / (N)	Y / (N)	T / –	0/10
Control Flow & Logic 30 pts total				
8. Conditional branch triggers on participant's answer	Y / (N)	Y / (N)	(T) / –	0/15
9. All four steps present in correct sequence	(Y) / N	(Y) / N	(T) / –	7.5/15

animated say

Raw Total: 42.5 / 100 **Tool-assisted items (T, for reference):** 7

Design Fidelity Score (DFS): 42.5 % *DFS = Raw Total. T marks are recorded only; they do not affect the score.*

Notes on Implementation Quality:

- No distinct movement blocks, only animated say.
- Needed assistance with manual branching

Overall Assessment — Did the wizard's design faithfully represent the specification?

Yes, the implemented design attempted to represent every element, and executed each. Execution, however, deviated.

Execution Reliability Score Rubric

HRIStudio Pilot Study – Completed by researcher after reviewing the video recording

Participant ID: W01 **Date:** 2026-04-01

Condition: ☐ HRIStudio ☒ Choregraphe

Trial duration: 5 min **Reached Step 4?** ☒ Y ☐ N **Subject's answer:** blue

Test subject: ☐ Recruited participant ☐ Researcher (note foreknowledge in session notes if Researcher)

Scoring: Full points if Executed *and* Correct *and* not Assisted. Half points if Executed but not Correct, *or* if Assisted (T). Zero if not Executed.

Assisted (T): Mark T only for *tool-operation* interventions during the live trial tied to this item. Design-phase T marks do *not* carry over from the DFS rubric. Do not mark T for hardware failures or general confusion unrelated to tool operation.

Gestures (items 5–7): Correct only if the gesture executed as a distinct, intentional robot movement. Incidental auto-motion during speech does *not* qualify — Animated Say selects movements non-deterministically, so output varies between runs even with an identical program.

Branching (item 4): Correct only if the branch resolved via programmed conditional logic. Manual re-routing or path selection by the wizard during the trial = Executed Y, Correct N.

Behavior	Executed?	Correctly?	Assisted?	Points
Speech Execution 40 pts total				
1. Introduction speech delivered without errors	(Y) / N	(Y) / N	T / –	10/10
2. Narrative speech delivered without errors	(Y) / N	(Y) / N	T / –	10/10
3. Comprehension question delivered correctly	(Y) / N	(Y) / N	T / –	10/10
4. Appropriate branch response given	(Y) / N	Y / (N)	(T) / –	5/10
Gesture & Movement Execution 30 pts total				
5. Introduction gesture (open-hand wave) executed	(Y) / N	Y / (N)	T / –	5/10
6. At least one narrative gesture executed	(Y) / N	Y / (N)	T / –	5/10
7. Nod or head shake executed correctly	Y / (N)	Y / (N)	T / –	0/10
Timing & Synchronization 20 pts total				
8. Speech and gestures synchronized	(Y) / N	(Y) / N	T / –	10/10
9. Pause held before comprehension question	Y / (N)	Y / (N)	T / –	0/10
System Reliability 10 pts – deduct if errors occur				
10. No disconnections, crashes, or hangs	(Y) / N	N/A	–	10/10

Raw Total: 65 / 100 **Trial-phase assisted (T count):** 4

Execution Reliability Score (ERS): 65 % *ERS = Raw Total. Assisted cap applied per-item above.*

Trial-Phase Intervention Log *Tool-type (T) marks reduce the relevant item's score above.*

Time	Type	ERS #	Description
3	T	4	When branching, researcher was asked how to jump branches manually.

Total trial interventions: 1 **Tool-type (T):** 1

Observed Errors or Issues:

Overall Assessment — Did the interaction execute as the wizard designed it?

Yes, executed as designed.

System Usability Scale (SUS)

HRIStudio Pilot Study – Completed by wizard after the live trial

Participant ID: 001 **Condition:** ☐ HRIStudio ☑ Choregraphe **Date:** 2026-04-01

For each statement below, circle the number that best reflects your agreement.
1 = Strongly Disagree 2 = Disagree 3 = Neutral 4 = Agree 5 = Strongly Agree

Statement	1	2	3	4	5	score
1. I think that I would like to use this system frequently.	○	○	○	⊗	○	3
2. I found the system unnecessarily complex.	○	⊗	○	○	○	3
3. I thought the system was easy to use.	○	○	⊗	○	○	2
4. I think I would need support from a technical person to use this system.	○	○	○	⊗	○	1
5. I found the various functions in this system were well integrated.	○	○	⊗	○	○	2
6. I thought there was too much inconsistency in this system.	⊗	○	○	○	○	4
7. I would imagine that most people would learn to use this system very quickly.	○	○	⊗	○	○	2
8. I found the system very cumbersome to use.	○	○	⊗	○	○	2
9. I felt very confident using the system.	○	○	⊗	○	○	2
10. I needed to learn a lot of things before I could get going with this system.	○	⊗	○	○	○	3

sum: 24
× 2.5
60

Scoring note (for researcher): Odd items are positive; even items are negative. Subtract 1 from each odd item's score; subtract each even item's score from 5. Sum all ten converted scores and multiply by 2.5. Score range: 0–100.

Observer Data Collection Sheet

HRIStudio Pilot Study – Completed by researcher during the live session

---

Participant ID: W-02 **Date:** 2026-04-07

Condition: ☒ HRIStudio ☐ Choregraphe

Session Start: 14:00 **Test ID:** 002 **Subject Room:** ACET 239

Test subject type: ☐ Recruited participant ☒ Researcher (foreknowledge — note in session notes)

Phase 1 – Training 15 min planned

Start: 14:03 **End:** 14:10 **Actual:** 7 min **Questions asked:** 0

Training notes:
Participant thought platform appeared to be straightforward.

Phase 2 – Design Challenge 30 min planned

Start: 14:10 **End:** 14:34 **Time to completion:** 24 min

Completed? ☒ Yes ☐ No (hit time limit)

Intervention log: *Type:* ***T*** *= tool operation* ***C*** *= task/spec clarification* ***H*** *= hardware/tech* ***G*** *= general/forgetfulness. T marks are recorded alongside the DFS; they do not affect the DFS score.*

Time	Type	DFS #	Description
14:11	T	N/A	Plugins were not added to the experiment automatically. Reused seeded study instead.
14:13	T	8	Observe block usage unclear; removed
14:16	T	8	Clarified that branch destinations are steps
14:20	C	N/A	Clarified paper spec is a generalization

Total interventions: 4 **Tool-type (T):** 3 **Other:** 1

Phase 3 – Live Trial 10 min planned

Start: 14:34 **End:** 14:39 **Actual:** 5 min **Reached Step 4?** ☒ Y ☐ N

Subject's answer: Red Branch triggered: ☐ A ☐ B ☒ None/manual

Researcher interventions during trial: *Same type codes as above. T-type interventions cap the relevant ERS item at 50%.*

Time	Type	ERS #	Description
14:36	T	4	Branch logic needed to be manually triggered.

Total trial interventions: 1 **Tool-type (T):** 1

Other issues during trial:

None

Phase 4 – Debrief 5 min planned

Start: 14:39 **End:** 14:44 **SUS completed?** ☒ Yes ☐ No

Project file exported? ☒ Yes ☐ No

Qualitative comments:

Done well. Detailed notes/descriptions added to steps/actions.

Overall Session Notes

Platform worked well, participant driven and curious.

Design Fidelity Score Rubric

HRIStudio Pilot Study – Completed by researcher after reviewing the exported project file

Participant ID: W-02 **Date:** 2026-04-07

Condition: ☒ HRIStudio ☐ Choregraphe

Scoring: Full points if Present *and* Correct. Half points if Present but not Correct. Zero if not Present.

Assisted (T): Mark T if a tool-operation intervention was given for this item. Recorded for reference — does *not* affect the Points total or the DFS.

Gestures (items 5–7): Correct only if gesture is a distinct action or animation node separate from the speech action. Auto-motion from "Animated Say" does *not* qualify — it selects movements non-deterministically, so output varies between runs even with the same program.

Branching (item 8): Correct only if a dedicated conditional control-flow node (Switch, Choice, or equivalent) is wired to a voice or input trigger. Manual re-routing during execution does *not* satisfy this criterion.

Component	Present?	Correct?	Assisted?	Points
Speech Actions 40 pts total				
1. Introduction: *"Hello. I want to tell you about Kai..."*	(Y) / N	(Y) / N	T / –	10/10
2. Narrative: *"Kai had been on the Martian surface for six days..."*	(Y) / N	(Y) / N	T / –	10/10
3. Question: *"What color was the rock Kai found?"*	(Y) / N	(Y) / N	T / –	10/10
4. Both branch responses (correct *and* incorrect)	(Y) / N	(Y) / N	T / –	10/10
Gestures & Actions 30 pts total				
5. Open-hand wave during introduction	(Y) / N	(Y) / N	T / –	10/10
6. At least one narrative gesture (pause or look down)	(Y) / N	(Y) / N	T / –	10/10
7. Nod (correct branch) or head shake (incorrect branch)	(Y) / N	(Y) / N	T / –	10/10
Control Flow & Logic 30 pts total				
8. Conditional branch triggers on participant's answer	(Y) / N	(Y) / N	T / –	15/15
9. All four steps present in correct sequence	(Y) / N	(Y) / N	T / –	15/15

Raw Total: 100 / 100 **Tool-assisted items (T, for reference):** 0

Design Fidelity Score (DFS): 100% *DFS = Raw Total. T marks are recorded only; they do not affect the score.*

Notes on Implementation Quality: Actions + order exact same as reference, Names of items modified.

Overall Assessment — Did the wizard's design faithfully represent the specification?

Yes, detailed notes, specific steps/actions mapped almost exactly to reference experiment. Functionally identical.

Execution Reliability Score Rubric

HRIStudio Pilot Study – Completed by researcher after reviewing the video recording

Participant ID: W02 **Date:** 2026-04-07

Condition: ☒ HRIStudio ☐ Choregraphe

Trial duration: 5 min **Reached Step 4?** ☒ Y ☐ N **Subject's answer:** Red

Test subject: ☐ Recruited participant ☒ Researcher (note foreknowledge in session notes if Researcher)

Scoring: Full points if Executed *and* Correct *and* not Assisted. Half points if Executed but not Correct, *or* if Assisted (T). Zero if not Executed.

Assisted (T): Mark T only for *tool-operation* interventions during the live trial tied to this item. Design-phase T marks do *not* carry over from the DFS rubric. Do not mark T for hardware failures or general confusion unrelated to tool operation.

Gestures (items 5–7): Correct only if the gesture executed as a distinct, intentional robot movement. Incidental auto-motion during speech does *not* qualify — Animated Say selects movements non-deterministically, so output varies between runs even with an identical program.

Branching (item 4): Correct only if the branch resolved via programmed conditional logic. Manual re-routing or path selection by the wizard during the trial = Executed Y, Correct N.

Behavior	Executed?	Correctly?	Assisted?	Points
Speech Execution 40 pts total				
1. Introduction speech delivered without errors	Y / N	Y / N	T / –	10/10
2. Narrative speech delivered without errors	Y / N	Y / N	T / –	10/10
3. Comprehension question delivered correctly	Y / N	Y / N	T / –	10/10
4. Appropriate branch response given	Y / N	Y / N	T / –	5/10
Gesture & Movement Execution 30 pts total				
5. Introduction gesture (open-hand wave) executed	Y / N	Y / N	T / –	10/10
6. At least one narrative gesture executed	Y / N	Y / N	T / –	10/10
7. Nod or head shake executed correctly	Y / N	Y / N	T / –	10/10
Timing & Synchronization 20 pts total				
8. Speech and gestures synchronized	Y / N	Y / N	T / –	10/10
9. Pause held before comprehension question	Y / N	Y / N	T / –	10/10
System Reliability 10 pts – deduct if errors occur				
10. No disconnections, crashes, or hangs	Y / N	N/A	–	10/10

Raw Total: 95 / 100 **Trial-phase assisted (T count):** 1

Execution Reliability Score (ERS): 95 % *ERS = Raw Total. Assisted cap applied per-item above.*

Trial-Phase Intervention Log *Tool-type (T) marks reduce the relevant item's score above.*

Time	Type	ERS #	Description
14:36	T	4	Branch logic needed to be manually triggered.

Total trial interventions: 0 **Tool-type (T):** 0

Observed Errors or Issues:

branching logic glitched platform-side - skipped manually.

Overall Assessment — Did the interaction execute as the wizard designed it?

Yes, other than branch glitch, execution was as designed.

System Usability Scale (SUS)

HRIStudio Pilot Study — Completed by wizard after the live trial

Participant ID: W82

Condition: ☒ HRIStudio ☐ Choregraphe **Date:** 2026-04-07

For each statement below, circle the number that best reflects your agreement.
1 = Strongly Disagree 2 = Disagree 3 = Neutral 4 = Agree 5 = Strongly Agree

Statement	1	2	3	4	5	
1. I think that I would like to use this system frequently.	○	○	○	⊘	○	3
2. I found the system unnecessarily complex.	⊘	○	○	○	○	4
3. I thought the system was easy to use.	○	○	○	⊘	○	3
4. I think I would need support from a technical person to use this system.	⊘	○	○	○	○	4
5. I found the various functions in this system were well integrated.	○	○	○	⊘	○	3
6. I thought there was too much inconsistency in this system.	⊘	○	○	○	○	4
7. I would imagine that most people would learn to use this system very quickly.	○	○	○	○	⊘	4
8. I found the system very cumbersome to use.	⊘	○	○	○	○	4
9. I felt very confident using the system.	○	○	○	⊘	○	3
10. I needed to learn a lot of things before I could get going with this system.	⊘	○	○	○	○	4
						90

Scoring note (for researcher): Odd items are positive; even items are negative. Subtract 1 from each odd item's score; subtract each even item's score from 5. Sum all ten converted scores and multiply by 2.5. Score range: 0–100.

Observer Data Collection Sheet

HRIStudio Pilot Study – Completed by researcher during the live session

Participant ID: W-03 **Date:** 2026-04-07

Condition: ☐ HRIStudio ☒ Choregraphe

Session Start: 15:00 **Test ID:** 003 **Subject Room:** ALET 239

Test subject type: ☐ Recruited participant ☒ Researcher (foreknowledge — note in session notes)

Phase 1 – Training 15 min planned

Start: 15:00 **End:** 15:12 **Actual:** 12 min **Questions asked:** 2

Training notes:

Asked about triggering branches
Asked about choice speech recognition block

Phase 2 – Design Challenge 30 min planned

Start: 15:12 **End:** 15:49 **Time to completion:** 37 min

Completed? ☒ Yes ☐ No (hit time limit)

Intervention log: *Type:* ***T*** *= tool operation* ***C*** *= task/spec clarification* ***H*** *= hardware/tech* ***G*** *= general/forgetfulness. T marks are recorded alongside the DFS; they do not affect the DFS score.*

Time	Type	DFS #	Description
15:22	T	6	Stopped participant from dangerously setting motor position.

Total interventions: 1 **Tool-type (T):** 1 **Other:** 0

Phase 3 – Live Trial 10 min planned

Start: 15:49 **End:** 15:54 **Actual:** 5 min **Reached Step 4?** ☒ Y ☐ N

Subject's answer: "No clue" Branch triggered: ☐ A ☐ B ☒ None/manual

Researcher interventions during trial: *Same type codes as above. T-type interventions cap the relevant ERS item at 50%.*

Time	Type	ERS #	Description

No interventions - wizard let experiment run, self-corrected on branching error by skipping to yes option

Total trial interventions: 0 **Tool-type (T):** 0

Other issues during trial:

Skipped block where color was mentioned in execution - participant was not asked. Comprehension was not possible

Phase 4 – Debrief 5 min planned

Start: 15:54 **End:** 15:59 **SUS completed?** ☒ Yes ☐ No

Project file exported? ☒ Yes ☐ No

Qualitative comments:

Lots of blocks present - wizard attempted to break spec up into smaller steps.

Overall Session Notes

Good session. Wizard liked to explore on their own - as expected given highly technical CS background.

Design Fidelity Score Rubric

HRIStudio Pilot Study – Completed by researcher after reviewing the exported project file

Participant ID: W-03 **Date:** 2026-04-07

Condition: ☐ HRIStudio ☒ Choregraphe

Scoring: Full points if Present *and* Correct. Half points if Present but not Correct. Zero if not Present.

Assisted (T): Mark T if a tool-operation intervention was given for this item. Recorded for reference — does *not* affect the Points total or the DFS.

Gestures (items 5–7): Correct only if gesture is a distinct action or animation node separate from the speech action. Auto-motion from "Animated Say" does *not* qualify — it selects movements non-deterministically, so output varies between runs even with the same program.

Branching (item 8): Correct only if a dedicated conditional control-flow node (Switch, Choice, or equivalent) is wired to a voice or input trigger. Manual re-routing during execution does *not* satisfy this criterion.

Component	Present?	Correct?	Assisted?	Points
Speech Actions 40 pts total				
1. Introduction: *"Hello. I want to tell you about Kai..."*	Y / N	Y / N	T / –	10/10
2. Narrative: *"Kai had been on the Martian surface for six days..."*	Y / N	Y / N	T / –	5/10
3. Question: *"What color was the rock Kai found?"*	Y / N	Y / N	T / –	10/10
4. Both branch responses (correct *and* incorrect)	Y / N	Y / N	T / –	5/10
Gestures & Actions 30 pts total				
5. Open-hand wave during introduction	Y / N	Y / N	T / –	5/10
6. At least one narrative gesture (pause or look down)	Y / N	Y / N	T / –	10/10
7. Nod (correct branch) or head shake (incorrect branch)	Y / N	Y / N	T / –	5/10
Control Flow & Logic 30 pts total				
8. Conditional branch triggers on participant's answer	Y / N	Y / N	T / –	0/15
9. All four steps present in correct sequence	Y / N	Y / N	T / –	15/15

Raw Total: 65 / 100 **Tool-assisted items (T, for reference):** 1

Design Fidelity Score (DFS): 65 % *DFS = Raw Total. T marks are recorded only; they do not affect the score.*

Notes on Implementation Quality:

Overall Assessment — Did the wizard's design faithfully represent the specification?

Yes, wizard's design, although complicated, attempted to implement all steps and actions.

Execution Reliability Score Rubric

HRIStudio Pilot Study – Completed by researcher after reviewing the video recording

Participant ID: W-03 **Date:** 2026-04-07

Condition: ☐ HRIStudio ☒ Choregraphe

Trial duration: 3 min **Reached Step 4?** ☒ Y ☐ N **Subject's answer:** "No clue"

Test subject: ☐ Recruited participant ☒ Researcher (note foreknowledge in session notes if Researcher)

Scoring: Full points if Executed *and* Correct *and* not Assisted. Half points if Executed but not Correct, *or* if Assisted (T). Zero if not Executed.

Assisted (T): Mark T only for *tool-operation* interventions during the live trial tied to this item. Design-phase T marks do *not* carry over from the DFS rubric. Do not mark T for hardware failures or general confusion unrelated to tool operation.

Gestures (items 5–7): Correct only if the gesture executed as a distinct, intentional robot movement. Incidental auto-motion during speech does *not* qualify — Animated Say selects movements non-deterministically, so output varies between runs even with an identical program.

Branching (item 4): Correct only if the branch resolved via programmed conditional logic. Manual re-routing or path selection by the wizard during the trial = Executed Y, Correct N.

Behavior	Executed?	Correctly?	Assisted?	Points
Speech Execution 40 pts total				
1. Introduction speech delivered without errors	(Y) / N	(Y) / N	T / –	10/10
2. Narrative speech delivered without errors	(Y) / N	Y / (N)	T / –	5/10
3. Comprehension question delivered correctly	(Y) / N	(Y) / N	T / –	10/10
4. Appropriate branch response given	(Y) / N	Y / (N)	T / –	5/10
Gesture & Movement Execution 30 pts total				
5. Introduction gesture (open-hand wave) executed	(Y) / N	Y / (N)	T / –	5/10
6. At least one narrative gesture executed	(Y) / N	Y / (N)	T / –	5/10
7. Nod or head shake executed correctly	Y / (N)	Y / (N)	T / –	0/10
Timing & Synchronization 20 pts total				
8. Speech and gestures synchronized	Y / (N)	Y / (N)	T / –	0/10
9. Pause held before comprehension question	(Y) / N	(Y) / N	T / –	10/10
System Reliability 10 pts – deduct if errors occur				
10. No disconnections, crashes, or hangs	(Y) / N	N/A	–	10/10

Raw Total: 60 / 100 **Trial-phase assisted (T count):** 0

Execution Reliability Score (ERS): 60 % *ERS = Raw Total. Assisted cap applied per-item above.*

Trial-Phase Intervention Log *Tool-type (T) marks reduce the relevant item's score above.*

Time	Type	ERS #	Description

No interventions - wizard let experiment run, self-corrected on branching error by skipping to yes option

Total trial interventions: 0 **Tool-type (T):** 0

Observed Errors or Issues:

Question not asked after story told - jumped to "yes" case manually.

Overall Assessment — Did the interaction execute as the wizard designed it?

No - wizard included question in design but left out a connection wire.

System Usability Scale (SUS)

HRIStudio Pilot Study – Completed by wizard after the live trial

Participant ID: W-03 **Condition:** ☐ HRIStudio ☒ Choregraphe **Date:** 2026-04-07

For each statement below, circle the number that best reflects your agreement.
1 = Strongly Disagree 2 = Disagree 3 = Neutral 4 = Agree 5 = Strongly Agree

Statement	1	2	3	4	5	
1. I think that I would like to use this system frequently.	○	○	○	●	○	3
2. I found the system unnecessarily complex.	●	○	○	○	○	4
3. I thought the system was easy to use.	○	○	○	●	○	3
4. I think I would need support from a technical person to use this system.	○	●	○	○	○	3
5. I found the various functions in this system were well integrated.	○	○	●	○	○	2
6. I thought there was too much inconsistency in this system.	○	●	○	○	○	3
7. I would imagine that most people would learn to use this system very quickly.	○	○	○	○	●	4
8. I found the system very cumbersome to use.	○	●	○	○	○	3
9. I felt very confident using the system.	○	○	●	○	○	2
10. I needed to learn a lot of things before I could get going with this system.	○	●	○	○	○	3
						+
						× 30 / 2.5 / 75

Scoring note (for researcher): Odd items are positive; even items are negative. Subtract 1 from each odd item's score; subtract each even item's score from 5. Sum all ten converted scores and multiply by 2.5. Score range: 0–100.

Observer Data Collection Sheet

HRIStudio Pilot Study – Completed by researcher during the live session

Participant ID: W-04 **Date:** 2026-04-08

Condition: ☐ HRIStudio ☒ Choregraphe

Session Start: 11:00 **Test ID:** 004 **Subject Room:** A239

Test subject type: ☐ Recruited participant ☒ Researcher (foreknowledge — note in session notes)

Phase 1 – Training 15 min planned

Start: 11:00 **End:** 11:17 **Actual:** 17 min **Questions asked:** 2

Training notes:

Asked about starting two blocks at the same time
Asked for clarification on editing a block

Phase 2 – Design Challenge 30 min planned

Start: 11:17 **End:** 11:52 **Time to completion:** 35 min

Completed? ☐ Yes ☒ No (hit time limit)

Intervention log: *Type:* ***T*** *= tool operation* ***C*** *= task/spec clarification* ***H*** *= hardware/tech* ***G*** *= general/forgetfulness. T marks are recorded alongside the DFS; they do not affect the DFS score.*

Time	Type	DFS #	Description
11:18	T x3	1-3	Asked about interpretation of punctuation
11:18	C	N/A	Asked about initial position of robot
11:46	T	8	Attempted to get choice block to work – it's bugged, I re-explained WoZ and spec/how to branch.

Total interventions: 5 **Tool-type (T):** 4 **Other:** 1

Phase 3 – Live Trial 10 min planned

Start: 11:52 **End:** 11:56 **Actual:** 4 min **Reached Step 4?** ☐ Y ☒ N

Subject's answer: N/A Branch triggered: ☐ A ☐ B ☒ None/manual

Researcher interventions during trial: *Same type codes as above. T-type interventions cap the relevant ERS item at 50%.*

Time	Type	ERS #	Description
11:54	T	4	Reminder of how to trigger branches

Total trial interventions: 1 **Tool-type (T):** 1

Other issues during trial:

No question asked - jumped to "yes" branch.

Phase 4 – Debrief 5 min planned

Start: 11:56 **End:** 12:00 **SUS completed?** ☒ Yes ☐ No

Project file exported? ☒ Yes ☐ No

Qualitative comments:

Quality attempted, but platform seemingly got in the way.

Overall Session Notes

Design Fidelity Score Rubric

HRIStudio Pilot Study – Completed by researcher after reviewing the exported project file

Participant ID: W-04 **Date:** 2026-04-08

Condition: ☐ HRIStudio ☒ Choregraphe

Scoring: Full points if Present *and* Correct. Half points if Present but not Correct. Zero if not Present.

Assisted (T): Mark T if a tool-operation intervention was given for this item. Recorded for reference — does *not* affect the Points total or the DFS.

Gestures (items 5–7): Correct only if gesture is a distinct action or animation node separate from the speech action. Auto-motion from "Animated Say" does *not* qualify — it selects movements non-deterministically, so output varies between runs even with the same program.

Branching (item 8): Correct only if a dedicated conditional control-flow node (Switch, Choice, or equivalent) is wired to a voice or input trigger. Manual re-routing during execution does *not* satisfy this criterion.

Component	Present?	Correct?	Assisted?	Points
Speech Actions 40 pts total				
1. Introduction: *"Hello. I want to tell you about Kai..."*	(Y) / N	(Y) / N	(T) / –	10/10
2. Narrative: *"Kai had been on the Martian surface for six days..."*	(Y) / N	(Y) / N	(T) / –	10/10
3. Question: *"What color was the rock Kai found?"*	Y / (N)	Y / (N)	(T) / –	0/10
4. Both branch responses (correct *and* incorrect)	(Y) / N	(Y) / N	T / –	10/10
Gestures & Actions 30 pts total				
5. Open-hand wave during introduction	(Y) / N	(Y) / N	T / –	10/10
6. At least one narrative gesture (pause or look down)	(Y) / N	(Y) / N	T / –	10/10
7. Nod (correct branch) or head shake (incorrect branch)	(Y) / N	Y / (N)	T / –	5/10
Control Flow & Logic 30 pts total				
8. Conditional branch triggers on participant's answer	Y / (N)	Y / (N)	(T) / –	0/15
9. All four steps present in correct sequence	(Y) / N	Y / (N)	T / –	7.5/15

Raw Total: 62.5 / 100 **Tool-assisted items (T, for reference):** 4

Design Fidelity Score (DFS): 62.5 % *DFS = Raw Total. T marks are recorded only; they do not affect the score.*

Notes on Implementation Quality:

Tried to use choice block but it didn't execute correctly. I had to explain WoZ/puppeteering

Overall Assessment — Did the wizard's design faithfully represent the specification?

Yes - attempted, but no branch broke execution.

Execution Reliability Score Rubric

HRIStudio Pilot Study – Completed by researcher after reviewing the video recording

Participant ID: W-04 **Date:** 2026-04-08

Condition: ☐ HRIStudio ☒ Choregraphe

Trial duration: 4 min **Reached Step 4?** ☐ Y ☐ N **Subject's answer:** _____

Test subject: ☐ Recruited participant ☒ Researcher (note foreknowledge in session notes if Researcher)

Scoring: Full points if Executed *and* Correct *and* not Assisted. Half points if Executed but not Correct, *or* if Assisted (T). Zero if not Executed.

Assisted (T): Mark T only for *tool-operation* interventions during the live trial tied to this item. Design-phase T marks do *not* carry over from the DFS rubric. Do not mark T for hardware failures or general confusion unrelated to tool operation.

Gestures (items 5–7): Correct only if the gesture executed as a distinct, intentional robot movement. Incidental auto-motion during speech does *not* qualify — Animated Say selects movements non-deterministically, so output varies between runs even with an identical program.

Branching (item 4): Correct only if the branch resolved via programmed conditional logic. Manual re-routing or path selection by the wizard during the trial = Executed Y, Correct N.

Behavior	Executed?	Correctly?	Assisted?	Points
Speech Execution 40 pts total				
1. Introduction speech delivered without errors	(Y) / N	(Y) / N	T / –	10/10
2. Narrative speech delivered without errors	(Y) / N	(Y) / N	T / –	10/10
3. Comprehension question delivered correctly	Y / (N)	Y / (N)	T / –	0/10
4. Appropriate branch response given	Y / (N)	Y / (N)	(T) / –	5/10
Gesture & Movement Execution 30 pts total				
5. Introduction gesture (open-hand wave) executed	(Y) / N	(Y) / N	T / –	10/10
6. At least one narrative gesture executed	(Y) / N	(Y) / N	T / –	10/10
7. Nod or head shake executed correctly	(Y) / N	(Y) / N	T / –	10/10
Timing & Synchronization 20 pts total				
8. Speech and gestures synchronized	(Y) / N	(Y) / N	T / –	10/10
9. Pause held before comprehension question	Y / (N)	Y / (N)	T / –	0/10
System Reliability 10 pts – deduct if errors occur				
10. No disconnections, crashes, or hangs	(Y) / N	N/A	–	10/10

Raw Total: 70 / 100 **Trial-phase assisted (T count):** 1

Execution Reliability Score (ERS): 70 % *ERS = Raw Total. Assisted cap applied per-item above.*

Trial-Phase Intervention Log *Tool-type (T) marks reduce the relevant item's score above.*

Time	Type	ERS #	Description
11:54	T	4	Reminder of how to trigger branches

Total trial interventions: 1 **Tool-type (T):** 1

Observed Errors or Issues:

Overall Assessment — Did the interaction execute as the wizard designed it?

Almost - wizard expected question to be asked. It was not.

System Usability Scale (SUS)

HRIStudio Pilot Study – Completed by wizard after the live trial

Participant ID: 04

Condition: ☐ HRIStudio ☒ Choregraphe **Date:** 4/8/26

For each statement below, circle the number that best reflects your agreement.
1 = Strongly Disagree 2 = Disagree 3 = Neutral 4 = Agree 5 = Strongly Agree

Statement	1	2	3	4	5
1. I think that I would like to use this system frequently.	○	○	●	○	○
2. I found the system unnecessarily complex.	○	○	○	●	○
3. I thought the system was easy to use.	○	○	●	○	○
4. I think I would need support from a technical person to use this system.	○	○	○	●	○
5. I found the various functions in this system were well integrated.	○	●	○	○	○
6. I thought there was too much inconsistency in this system.	○	○	○	●	○
7. I would imagine that most people would learn to use this system very quickly.	○	○	●	○	○
8. I found the system very cumbersome to use.	○	○	●	○	○
9. I felt very confident using the system.	○	○	●	○	○
10. I needed to learn a lot of things before I could get going with this system.	○	●	○	○	○

Scoring note (for researcher): Odd items are positive; even items are negative. Subtract 1 from each odd item's score; subtract each even item's score from 5. Sum all ten converted scores and multiply by 2.5. Score range: 0–100.

Observer Data Collection Sheet

HRIStudio Pilot Study – Completed by researcher during the live session

Participant ID: W-05 **Date:** 2026-04-08

Condition: ☒ HRIStudio ☐ Choregraphe

Session Start: 15:30 **Test ID:** 005 **Subject Room:** DANA 327

Test subject type: ☐ Recruited participant ☒ Researcher (foreknowledge — note in session notes)

Phase 1 – Training 15 min planned

Start: 15:40 **End:** 15:46 **Actual:** 6 min **Questions asked:** 0

Training notes:

Saw platform as pretty straightforward.

Phase 2 – Design Challenge 30 min planned

Start: 15:46 **End:** 16:04 crouch bugged **Time to completion:** 18 min

Completed? ☒ Yes ☐ No (hit time limit)

Intervention log: *Type: **T** = tool operation **C** = task/spec clarification **H** = hardware/tech **G** = general/forgetfulness. T marks are recorded alongside the DFS; they do not affect the DFS score.*

Time	Type	DFS #	Description
15:48	T	1	Clarified that properties can be edited on right pane of experiment designer
15:50	C	N/A	Clarified that "steps" are arbitrary containers
15:53	C	..	..
15:55	C	N/A	Clarified that spec "speech" doesn't specifically map to speech actions
15:56	T	4	Clarified that branch block requires predefined steps

Total interventions: 5 **Tool-type (T):** 2 **Other:** 3

Phase 3 – Live Trial 10 min planned

Start: 16:04 **End:** 16:08 **Actual:** 4 min **Reached Step 4?** ☒ Y ☐ N

Subject's answer: Red Branch triggered: ☒ A ☐ B ☐ None/manual

Researcher interventions during trial: *Same type codes as above. T-type interventions cap the relevant ERS item at 50%.*

Time	Type	ERS #	Description
16:04	G		Researcher forgot they were live executing
16:05	T	6	skipped over non-functional crouch.

Total trial interventions: 2 **Tool-type (T):** 1

Other issues during trial:
Crouch action exists but does not execute robot-side. It was skipped.

Phase 4 – Debrief 5 min planned

Start: 16:08 **End:** 16:12 **SUS completed?** ☒ Yes ☐ No

Project file exported? ☒ Yes ☐ No

Qualitative comments: No issues.

Overall Session Notes

Overall good session. Learning different backgrounds determine tool curiosity and drive to self-explore.

Design Fidelity Score Rubric

HRIStudio Pilot Study – Completed by researcher after reviewing the exported project file

Participant ID: W-05 **Date:** 2026-04-08

Condition: ☒ HRIStudio ☐ Choregraphe

Scoring: Full points if Present *and* Correct. Half points if Present but not Correct. Zero if not Present.

Assisted (T): Mark T if a tool-operation intervention was given for this item. Recorded for reference — does *not* affect the Points total or the DFS.

Gestures (items 5–7): Correct only if gesture is a distinct action or animation node separate from the speech action. Auto-motion from "Animated Say" does *not* qualify — it selects movements non-deterministically, so output varies between runs even with the same program.

Branching (item 8): Correct only if a dedicated conditional control-flow node (Switch, Choice, or equivalent) is wired to a voice or input trigger. Manual re-routing during execution does *not* satisfy this criterion.

Component	Present?	Correct?	Assisted?	Points
Speech Actions 40 pts total				
1. Introduction: *"Hello. I want to tell you about Kai..."*	(Y) / N	(Y) / N	(T) / –	10/10
2. Narrative: *"Kai had been on the Martian surface for six days..."*	(Y) / N	(Y) / N	T / –	10/10
3. Question: *"What color was the rock Kai found?"*	(Y) / N	(Y) / N	T / –	10/10
4. Both branch responses (correct *and* incorrect)	(Y) / N	(Y) / N	(T) / –	10/10
Gestures & Actions 30 pts total				
5. Open-hand wave during introduction	(Y) / N	(Y) / N	T / –	10/10
6. At least one narrative gesture (pause or look down)	(Y) / N	(Y) / N	T / –	10/10
7. Nod (correct branch) or head shake (incorrect branch)	(Y) / N	(Y) / N	T / –	10/10
Control Flow & Logic 30 pts total				
8. Conditional branch triggers on participant's answer	(Y) / N	(Y) / N	T / –	15/15
9. All four steps present in correct sequence	(Y) / N	(Y) / N	T / –	15/15

Raw Total: 100 / 100 **Tool-assisted items (T, for reference):** 2

Design Fidelity Score (DFS): 100% *DFS = Raw Total. T marks are recorded only; they do not affect the score.*

Notes on Implementation Quality: Done well, no comments or edits to step/action names though.

Overall Assessment — Did the wizard's design faithfully represent the specification?

Yes. Wizard's design mapped well from spec.

Execution Reliability Score Rubric

HRIStudio Pilot Study – Completed by researcher after reviewing the video recording

Participant ID: W-05 **Date:** 2026-04-08

Condition: ☒ HRIStudio ☐ Choregraphe

Trial duration: 4 min **Reached Step 4?** ☒ Y ☐ N **Subject's answer:** RED

Test subject: ☐ Recruited participant ☒ Researcher (note foreknowledge in session notes if Researcher)

Scoring: Full points if Executed *and* Correct *and* not Assisted. Half points if Executed but not Correct, *or* if Assisted (T). Zero if not Executed.

Assisted (T): Mark T only for *tool-operation* interventions during the live trial tied to this item. Design-phase T marks do *not* carry over from the DFS rubric. Do not mark T for hardware failures or general confusion unrelated to tool operation.

Gestures (items 5–7): Correct only if the gesture executed as a distinct, intentional robot movement. Incidental auto-motion during speech does *not* qualify — Animated Say selects movements non-deterministically, so output varies between runs even with an identical program.

Branching (item 4): Correct only if the branch resolved via programmed conditional logic. Manual re-routing or path selection by the wizard during the trial = Executed Y, Correct N.

Behavior	Executed?	Correctly?	Assisted?	Points
Speech Execution 40 pts total				
1. Introduction speech delivered without errors	(Y) / N	(Y) / N	T / –	10/10
2. Narrative speech delivered without errors	(Y) / N	(Y) / N	T / –	10/10
3. Comprehension question delivered correctly	(Y) / N	(Y) / N	T / –	10/10
4. Appropriate branch response given	(Y) / N	(Y) / N	T / –	10/10
Gesture & Movement Execution 30 pts total				
5. Introduction gesture (open-hand wave) executed	(Y) / N	(Y) / N	T / –	10/10
6. At least one narrative gesture executed	(Y) / N	(Y) / N	(T) / –	5/10
7. Nod or head shake executed correctly	(Y) / N	(Y) / N	T / –	10/10
Timing & Synchronization 20 pts total				
8. Speech and gestures synchronized	(Y) / N	(Y) / N	T / –	10/10
9. Pause held before comprehension question	(Y) / N	(Y) / N	T / –	10/10
System Reliability 10 pts – deduct if errors occur				
10. No disconnections, crashes, or hangs	(Y) / N	N/A	–	10/10

Raw Total: 95 / 100 **Trial-phase assisted (T count):** 0

Execution Reliability Score (ERS): 95 % *ERS = Raw Total. Assisted cap applied per-item above.*

Trial-Phase Intervention Log *Tool-type (T) marks reduce the relevant item's score above.*

Time	Type	ERS #	Description
16:04	G	N/A	Researcher forgot they were live executing
16:05	T	6	skipped over non-functional crouch.

Total trial interventions: 2 **Tool-type (T):** 1

Observed Errors or Issues:

Crouch action exists but does not execute robot-side. It was skipped.

Overall Assessment — Did the interaction execute as the wizard designed it?

Yes, interaction executed as designed.

System Usability Scale (SUS)

HRIStudio Pilot Study — Completed by wizard after the live trial

Participant ID: W-05 **Condition:** ☒ HRIStudio ☐ Choreographe **Date:** 2026-04-08

For each statement below, circle the number that best reflects your agreement.
1 = Strongly Disagree 2 = Disagree 3 = Neutral 4 = Agree 5 = Strongly Agree

Statement	1	2	3	4	5	
1. I think that I would like to use this system frequently.	○	○	○	✓	○	3
2. I found the system unnecessarily complex.	○	✓	○	○	○	3
3. I thought the system was easy to use.	○	○	○	✓	○	3
4. I think I would need support from a technical person to use this system.	○	○	✓	○	○	2
5. I found the various functions in this system were well integrated.	○	○	○	✓	○	3
6. I thought there was too much inconsistency in this system.	○	○	✓	○	○	2
7. I would imagine that most people would learn to use this system very quickly.	○	○	○	✓	○	3
8. I found the system very cumbersome to use.	○	✓	○	○	○	3
9. I felt very confident using the system.	○	○	○	✓	○	3
10. I needed to learn a lot of things before I could get going with this system.	○	✓	○	○	○	3
						+ 3
						× 28
						70

Scoring note (for researcher): Odd items are positive; even items are negative. Subtract 1 from each odd item's score; subtract each even item's score from 5. Sum all ten converted scores and multiply by 2.5. Score range: 0–100.

Observer Data Collection Sheet

HRIStudio Pilot Study – Completed by researcher during the live session

Participant ID: W-06 **Date:** 2026-04-10

Condition: ☒ HRIStudio ☐ Choregraphe

Session Start: 15:00 **Test ID:** 006 **Subject Room:** ACET 239

Test subject type: ☐ Recruited participant ☒ Researcher (foreknowledge — note in session notes)

Phase 1 – Training 15 min planned

Start: 15:01 **End:** 15:09 **Actual:** 8 min **Questions asked:** 0

Training notes:

Straightforward training.

Phase 2 – Design Challenge 30 min planned

Start: 15:09 **End:** 15:30 **Time to completion:** 21 min

Completed? ☒ Yes ☐ No (hit time limit)

Intervention log: *Type:* ***T*** *= tool operation* ***C*** *= task/spec clarification* ***H*** *= hardware/tech* ***G*** *= general/forgetfulness. T marks are recorded alongside the DFS; they do not affect the DFS score.*

Time	Type	DFS #	Description
15:21	T	6	Attempted to use parallel execution, unable to edit action.
15:24	T	6	Trouble resetting posture, recommended blocks.

Total interventions: 2 **Tool-type (T):** 2 **Other:** 0

Phase 3 – Live Trial 10 min planned

Start: 15:30 **End:** 15:33 **Actual:** 3 min **Reached Step 4?** ☑ Y ☐ N

Subject's answer: Red Branch triggered: ☒ A ☐ B ☐ None/manual

Researcher interventions during trial: *Same type codes as above. T-type interventions cap the relevant ERS item at 50%.*

Time	Type	ERS #	Description
N/A			

Total trial interventions: 0 **Tool-type (T):** 0

Other issues during trial:

Phase 4 – Debrief 5 min planned

Start: 15:32 **End:** 15:38 **SUS completed?** ☒ Yes ☐ No

Project file exported? ☒ Yes ☐ No

Qualitative comments:

Great implementation. Attempted to overcomplicate, but corrected on testing.

Overall Session Notes

Good session.

Design Fidelity Score Rubric

HRIStudio Pilot Study – Completed by researcher after reviewing the exported project file

Participant ID: W-06 **Date:** 2026-04-10

Condition: ☒ HRIStudio ☐ Choregraphe

Scoring: Full points if Present *and* Correct. Half points if Present but not Correct. Zero if not Present.

Assisted (T): Mark T if a tool-operation intervention was given for this item. Recorded for reference — does *not* affect the Points total or the DFS.

Gestures (items 5–7): Correct only if gesture is a distinct action or animation node separate from the speech action. Auto-motion from "Animated Say" does *not* qualify — it selects movements non-deterministically, so output varies between runs even with the same program.

Branching (item 8): Correct only if a dedicated conditional control-flow node (Switch, Choice, or equivalent) is wired to a voice or input trigger. Manual re-routing during execution does *not* satisfy this criterion.

Component	Present?	Correct?	Assisted?	Points
Speech Actions 40 pts total				
1. Introduction: *"Hello. I want to tell you about Kai..."*	Y / N	Y / N	T / –	10/10
2. Narrative: *"Kai had been on the Martian surface for six days..."*	Y / N	Y / N	T / –	10/10
3. Question: *"What color was the rock Kai found?"*	Y / N	Y / N	T / –	10/10
4. Both branch responses (correct *and* incorrect)	Y / N	Y / N	T / –	10/10
Gestures & Actions 30 pts total				
5. Open-hand wave during introduction	Y / N	Y / N	T / –	10/10
6. At least one narrative gesture (pause or look down)	Y / N	Y / N	T/x2	10/10
7. Nod (correct branch) or head shake (incorrect branch)	Y / N	Y / N	T / –	10/10
Control Flow & Logic 30 pts total				
8. Conditional branch triggers on participant's answer	Y / N	Y / N	T / –	15/15
9. All four steps present in correct sequence	Y / N	Y / N	T / –	15/15

Raw Total: 100 / 100 **Tool-assisted items (T, for reference):** 2

Design Fidelity Score (DFS): 100 % *DFS = Raw Total. T marks are recorded only; they do not affect the score.*

Notes on Implementation Quality:

Overall Assessment — Did the wizard's design faithfully represent the specification?

Yes. Extra actions existed, but spec was hit. Extras were for posture reset.

Execution Reliability Score Rubric

HRIStudio Pilot Study – Completed by researcher after reviewing the video recording

Participant ID: W-06 **Date:** 2026-04-10

Condition: ☒ HRIStudio ☐ Choregraphe

Trial duration: 3 min **Reached Step 4?** ☒ Y ☐ N **Subject's answer:** Red

Test subject: ☐ Recruited participant ☒ Researcher (note foreknowledge in session notes if Researcher)

Scoring: Full points if Executed *and* Correct *and* not Assisted. Half points if Executed but not Correct, *or* if Assisted (T). Zero if not Executed.

Assisted (T): Mark T only for *tool-operation* interventions during the live trial tied to this item. Design-phase T marks do *not* carry over from the DFS rubric. Do not mark T for hardware failures or general confusion unrelated to tool operation.

Gestures (items 5–7): Correct only if the gesture executed as a distinct, intentional robot movement. Incidental auto-motion during speech does *not* qualify — Animated Say selects movements non-deterministically, so output varies between runs even with an identical program.

Branching (item 4): Correct only if the branch resolved via programmed conditional logic. Manual re-routing or path selection by the wizard during the trial = Executed Y, Correct N.

Behavior	Executed?	Correctly?	Assisted?	Points
Speech Execution 40 pts total				
1. Introduction speech delivered without errors	Y / N	Y / N	T / –	10/10
2. Narrative speech delivered without errors	Y / N	Y / N	T / –	10/10
3. Comprehension question delivered correctly	Y / N	Y / N	T / –	10/10
4. Appropriate branch response given	Y / N	Y / N	T / –	10/10
Gesture & Movement Execution 30 pts total				
5. Introduction gesture (open-hand wave) executed	Y / N	Y / N	T / –	10/10
6. At least one narrative gesture executed	Y / N	Y / N	T / –	10/10
7. Nod or head shake executed correctly	Y / N	Y / N	T / –	10/10
Timing & Synchronization 20 pts total				
8. Speech and gestures synchronized	Y / N	Y / N	T / –	10/10
9. Pause held before comprehension question	Y / N	Y / N	T / –	10/10
System Reliability 10 pts – deduct if errors occur				
10. No disconnections, crashes, or hangs	Y / N	N/A	–	10/10

Raw Total: 102 / 100 **Trial-phase assisted (T count):** 0

Execution Reliability Score (ERS): 100 % *ERS = Raw Total. Assisted cap applied per-item above.*

Trial-Phase Intervention Log *Tool-type (T) marks reduce the relevant item's score above.*

Time	Type	ERS #	Description
N/A			

Total trial interventions: 0 **Tool-type (T):** 0

Observed Errors or Issues:

None

Overall Assessment — Did the interaction execute as the wizard designed it?

Yes!

System Usability Scale (SUS)

HRIStudio Pilot Study – Completed by wizard after the live trial

Participant ID: W-06 **Condition:** ☒ HRIStudio ☐ Choregraphe **Date:** 2026-04-10

For each statement below, circle the number that best reflects your agreement.
1 = Strongly Disagree 2 = Disagree 3 = Neutral 4 = Agree 5 = Strongly Agree

Statement	1	2	3	4	5	
1. I think that I would like to use this system frequently.	○	○	○	◉	○	3
2. I found the system unnecessarily complex.	◉	○	○	○	○	4
3. I thought the system was easy to use.	○	○	○	◉	○	3
4. I think I would need support from a technical person to use this system.	○	◉	○	○	○	3
5. I found the various functions in this system were well integrated.	○	○	○	◉	○	3
6. I thought there was too much inconsistency in this system.	◉	○	○	○	○	4
7. I would imagine that most people would learn to use this system very quickly.	○	○	○	◉	○	3
8. I found the system very cumbersome to use.	○	◉	○	○	○	3
9. I felt very confident using the system.	○	◉	◉	○	○	2
10. I needed to learn a lot of things before I could get going with this system.	○	◉	○	○	○	+ 3
						28
						× 2.5
						70

Scoring note (for researcher): Odd items are positive; even items are negative. Subtract 1 from each odd item's score; subtract each even item's score from 5. Sum all ten converted scores and multiply by 2.5. Score range: 0–100.

Appendix C

Technical Documentation

This appendix documents the specific technologies, infrastructure, and integration mechanisms used to build HRIStudio, organized by the three architectural layers described in Chapter 4. The goal here is reference, not justification; Chapter 5 explains the reasoning behind the major architectural choices.

C.1 Technology Stack

Table C.1 lists the principal dependencies and their roles. The entire codebase is written in TypeScript, so type inconsistencies between layers are caught at compile time rather than appearing as runtime failures during a trial.

Component	Version	Role
Next.js (App Router)	16.2	Full-stack React framework
React	19.2	User interface rendering
TypeScript	—	Static typing across the full stack
tRPC	11.10	Type-safe API between client and server
Better Auth	1.5	Authentication and session management
Drizzle ORM	0.41	Type-safe database access and migrations
PostgreSQL	15	Primary relational database
MinIO	latest	S3-compatible object storage (video/audio)
Bun	runtime	WebSocket server for real-time trial communication
Tailwind CSS + shadcn/ui	4.1 / 0.0.4	Styling and UI component library
`@dnd-kit`	—	Drag-and-drop for experiment designer
ROS 2 Humble	—	Robot middleware (NAO6 integration stack)
Docker Compose	—	Multi-container orchestration

Table C.1: Principal dependencies in the HRIStudio technology stack.

C.1.1 User Interface Layer

The frontend is built on Next.js using React and TypeScript. Styling is handled with Tailwind CSS and the shadcn/ui component library, which provides accessible, pre-built UI primitives built on Radix UI. The drag-and-drop canvas in the Design interface uses the `@dnd-kit` library (`@dnd-kit/core` and `@dnd-kit/sortable`) to manage nested drag operations for arranging steps and action blocks.

C.1.2 Application Logic Layer

The server runs as a Next.js process. API routes use tRPC over HTTP for typed request/response calls; real-time communication during live trials uses a separate WebSocket server running on the Bun runtime (described in Section C.3). Authentication and session management are handled by Better Auth with the Drizzle adapter for database-backed sessions. Passwords are hashed with bcrypt (cost factor 12). Currently, credential-based (username and password) authentication is supported; the

architecture allows adding OAuth providers without changes to the session model.

C.1.3 Data and Robot Control Layer

Experiment protocols, trial records, and user data are stored in PostgreSQL. The schema and all database queries are managed through Drizzle ORM, which provides compile-time type safety for database interactions. Action configuration parameters and plugin-specific fields are stored as JSONB columns, which allows the same schema to accommodate any robot's action types without schema migrations.

Video and audio recordings captured during trials are stored in a self-hosted MinIO instance, an S3-compatible object storage service. Recordings are captured in the browser using the native MediaRecorder API and uploaded to MinIO when the trial concludes. Structured data (experiment specifications, trial event logs) and media files are stored separately: the database handles queryable records, and MinIO handles large binary files that the system never queries by content.

Robot communication is handled through a ROS 2 WebSocket bridge running on the robot's local network. The HRIStudio server connects to the bridge over a WebSocket and exchanges JSON-encoded ROS messages; it does not run as a ROS node itself. The bridge address is configured per robot in the plugin file. For actions that do not require ROS message passing, the system can also execute commands directly on the robot via SSH (see Section C.2.2).

C.2 Deployment Infrastructure

HRIStudio uses a double Docker Compose stack: one stack runs the application and its backing services, and a second stack runs the robot integration layer. This separation allows the application to run on any host while the robot stack runs on a machine with network access to the physical robot. Both stacks can run on the same machine for single-lab deployments.

C.2.1 Application Stack

The application stack is defined in `hristudio/docker-compose.yml` and provides three services:

db. PostgreSQL 15 with a persistent named volume. Exposes port 5432.

minio. MinIO object storage with a persistent named volume. Exposes port 9000 (S3 API) and port 9001 (web console).

createbuckets. An initialization container that runs once at startup using the MinIO client to create the default storage bucket.

The Next.js application server and the Bun WebSocket server run outside Docker on the host, connecting to the containerized database and object store. Starting the backing services requires a single `docker compose up` command. This configuration is intended for on-premises deployment, which is important for studies involving participant data that cannot leave the institution's network.

C.2.2 NAO6 Integration Stack

The NAO6 integration stack is defined in a separate repository and provides three ROS 2 services that collectively bridge HRIStudio to the physical robot.

1. The **nao_driver** service runs the NAOqi driver ROS 2 node, which connects to the NAO's proprietary framework over the local network and publishes sensor data (joint states, camera feeds) as standard ROS 2 topics.
2. The **ros_bridge** service runs the `rosbridge` WebSocket server, which exposes all ROS 2 topics over a WebSocket interface on a configurable port (default 9090). This is the endpoint that the HRIStudio server connects to.
3. The **ros_api** service provides runtime introspection of available ROS 2 topics, services, and parameters.

All three services are built from a single Dockerfile based on the ROS 2 Humble base image (Ubuntu 22.04). The image installs the NAOqi driver and `rosbridge` server packages along with their dependencies (NAOqi libraries, bridge message types, OpenCV bridge, and TF2) and builds them with colcon. All services use host networking so that ROS 2 discovery and the NAOqi connection operate without port forwarding.

Before starting the driver, an initialization script connects to the NAO via SSH and prepares it for external control:

1. Disables Autonomous Life, which would otherwise cause the robot to move unpredictably.

2. Calls `ALMotion.wakeUp` to energize the motors.
3. Commands the robot to assume a standing posture via the ALRobotPosture service.

Environment variables for the robot IP address, credentials, and bridge port are read from a `.env` file shared across all three services.

C.2.3 Communication Between Stacks

Figure C.1 shows the relationship between the two Docker stacks and the components that run on the host. The HRIStudio server communicates with the robot integration stack over a single WebSocket connection to the `rosbridge_websocket` endpoint. For actions that bypass ROS entirely (posture changes, animation playback), the server connects directly to the NAO via SSH and invokes NAOqi commands through the `qicli` command-line tool. Both communication paths are configured per-robot in the plugin file.

C.3 WebSocket Architecture

Real-time communication during trials is handled by a dedicated WebSocket server that runs as a separate process alongside the Next.js application server. The WebSocket server is implemented in TypeScript and runs on the Bun runtime, listening on port 3001.

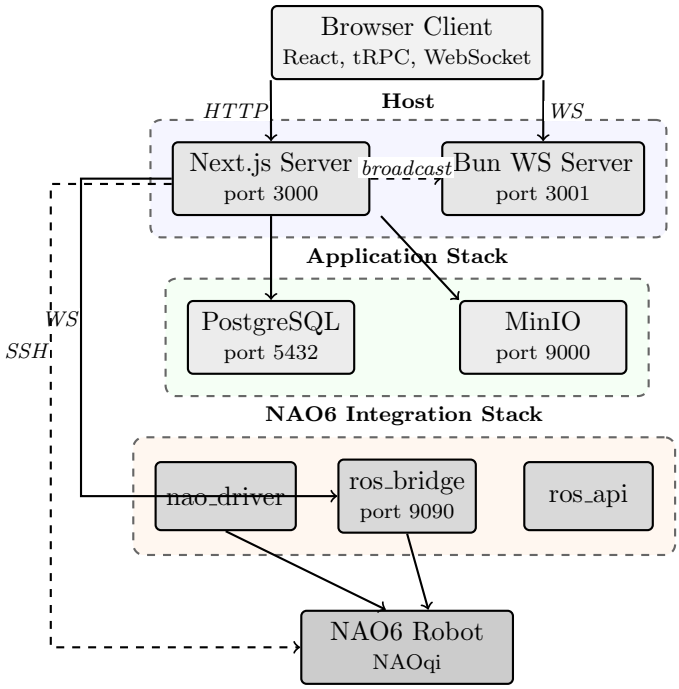


Figure C.1: Deployment architecture: two Docker stacks and their communication paths.

When a wizard or observer opens the Execution interface for a trial, the browser establishes a WebSocket connection to the server, passing the trial identifier and an authentication token as query parameters. The server registers the connection in an in-memory map keyed by client identifier and also records it in the database (`hs_ws_connection` table) for persistence across restarts.

The server handles four message types from connected clients:

Heartbeat. Keeps the connection alive; the server responds with a timestamped acknowledgment.

Request trial status. Returns the current trial state (status, current step index) by querying the database.

Request trial events. Returns the most recent trial events from the trial event log

table.

Ping. Returns a pong response with a timestamp for latency measurement.

When the Next.js server needs to push an update to all clients observing a trial (for example, after a step completes), it sends an HTTP POST to the WebSocket server's internal `/internal/broadcast` endpoint. The WebSocket server then forwards the message to every client registered for that trial. This architecture separates the stateful WebSocket connections from the stateless HTTP request handling of the Next.js server.

C.4 Plugin System

Robot capabilities are defined in JSON plugin files hosted in a plugin repository. A plugin repository is a static file server (served by an nginx container on port 8080 in the default configuration) that exposes three resources:

`repository.json`. Repository metadata including name, maintainers, trust level, supported ROS 2 distributions, and compatibility constraints.

`plugins/index.json`. An array of plugin filenames available in the repository.

`plugins/{name}.json`. Individual plugin files, one per robot platform.

When an administrator triggers a repository sync in the HRIStudio admin interface, the server fetches the repository metadata, retrieves the plugin index, and then

fetches each plugin file. The action definitions from each plugin are stored as JSONB in the `hs_robot_plugin` database table, making them available to the experiment designer and the execution engine without further network requests.

C.4.1 Plugin File Structure

Each plugin file is a self-contained description of a robot platform. The top-level fields include robot metadata (name, manufacturer, version, capabilities, physical specifications), a ROS 2 configuration block (namespace, default topics), and an array of action definitions. The official repository currently contains three plugins: `nao6-ros2.json`, `turtlebot3-burger.json`, and `turtlebot3-waffle.json`.

Each action definition specifies:

- A unique identifier (e.g., `say_text`, `walk_forward`, `play_animation_bow`).
- A human-readable name and icon for display in the Design interface.
- A parameter schema (JSON Schema format) defining the input fields the researcher configures.
- A timeout and retry policy.
- A ROS 2 dispatch block containing the target topic, message type, and a payload mapping.

The payload mapping supports two modes. In *static* mode, the plugin defines a fixed message template with placeholder tokens (e.g., {{`text`}}) that the execution

engine fills from the researcher's parameters. In *SSH* mode, the action bypasses ROS entirely and executes a shell command on the robot via SSH; this is used for NAOqi-native operations such as posture changes and animation playback that are not exposed as ROS 2 topics.

The NAO6 plugin defines 20 actions across three categories: speech (say text, say with emotion), movement (walk forward/backward, turn, stop, wake up, rest, stand, sit, crouch), and animation (bow, wave, nod, head shake, shrug, enthusiastic gesture, and others). Movement actions publish ROS 2 Twist messages to the velocity command topic. Animation actions publish animation path strings to the animation topic. Posture and lifecycle commands use SSH mode to call NAOqi services directly via the `qicli` command-line tool.

C.4.2 Adding a New Robot

Adding support for a new robot platform requires writing a single JSON plugin file and placing it in the plugin repository. No changes to the HRIStudio server code are required. The plugin author defines the robot's capabilities, maps each action to a ROS 2 topic or SSH command, and specifies the parameter schema for each action. After the repository is synced, the new robot's actions appear in the experiment designer and can be used in any study.

C.5 Database Schema

The database schema is managed through Drizzle ORM and uses a consistent `hs_` prefix for all tables. The schema is organized into five groups:

Authentication. User accounts, sessions, and system role assignments.

Study management. Studies with status tracking, study membership with per-study roles, and participant records with consent tracking.

Experimental design. Experiments, steps, and actions. Each action stores its transport type, configuration, parameter schema, and retry policy as JSONB columns.

Trial execution. Trials with status and duration tracking, and a trial event log that records every action, step transition, and deviation with a timestamp.

Robot integration. Robot definitions and installed plugins with cached action definitions. A block registry maps visual blocks in the experiment designer to their underlying action types, parameter schemas, and display properties.

All tables use a consistent `hs_` prefix (e.g., `hs_study`, `hs_trial`, `hs_action`).

C.6 Role-Based Access Control

As described in Chapter 5, HRIStudio uses a two-layer role system. System roles are stored in the `systemRoleEnum` column: *administrator*, *researcher*, *wizard*, and

observer. Study roles are stored in the `studyMemberRoleEnum` column: *owner*, *researcher*, *wizard*, and *observer*. The two layers are checked independently at the database level. On the server, tRPC middleware enforces access control: public procedures require no authentication, protected procedures require a valid session, and admin procedures additionally verify the user's system role. Study-level permissions are checked per-request by querying the `hs_study_member` table.

Appendix D

AI-Assisted Development Workflow

This appendix documents the role that AI coding assistants played in the construction of HRIStudio. It is included both for transparency about how the system was built and because the workflow itself is, in my view, one of the more interesting artifacts produced by the project. Section 5.1.1 in Chapter 5 introduces the topic briefly; here I describe the specific responsibilities I kept for myself, the tasks I delegated to coding agents, the tools I used, the limits I encountered, and the integrity controls I maintained between implementation work and the evaluation reported in Chapter 7.

D.1 Context

I built HRIStudio while also carrying a full course load, writing this thesis, and running the pilot validation study described in Chapter 6. The feature surface described in Chapters 4 and 5 is larger than what I could reasonably have produced on that schedule without assistance, given both the scope and the level of ambition of the work. AI coding assistants made that scope tractable. They did not replace design judgment; they reduced the cost of the mechanical work that sits between a well-specified design and a working feature: scaffolding new modules, implementing well-defined create/read/update/delete (CRUD) and validation code, applying consistent patterns across files, and producing the many small edits that a project of this size accumulates.

The set of tools available to me as a solo developer changed substantially during the project's timeline. When I began, agentic coding tools were still early and most of my AI use was conversational, primarily through Cursor [30] and Zed [24]. By the end of the project, multiple mature terminal- and editor-integrated agents were available. I changed tools as the landscape evolved, eventually moving into a mixed workflow between Visual Studio Code, Antigravity [23], Claude Code [20], and OpenCode [21].

D.2 Tools and Hardware

Table D.1 lists the tools I used during development and the capacity in which I used each. The split between them was determined partly by capability and partly by availability over time.

Tool	Category	Primary use
Claude [19]	Chat model	Design discussions, architectural review, debugging assistance, and refactoring proposals.
Claude Code [20]	Terminal agent	Multi-file feature implementation against a written spec; codemod-style refactors; and test scaffolding.
OpenCode [21]	Terminal agent	Same class of task as Claude Code, used when I preferred an open-source workflow or a different backing model.
Gemini CLI [22]	Terminal agent	Occasional cross-check on changes produced by a different agent, and work against Google's models when I wanted a second reading of a larger diff.
Antigravity [23]	IDE agent	Editor-integrated agentic coding work, primarily late in the project as the tool became available.
Cursor [30]	Editor	Early development; AI-native editing and indexing.
Zed [24]	Editor	High-performance editing; transition phase before moving to specialized agents.

Table D.1: AI tools used during HRIStudio development.

Beyond cloud-hosted models, I experimented with local execution using `llama.cpp` to run various open-weights models on my local hardware (Apple M4 Pro, 14-core CPU, 48GB RAM). While the hardware was capable of running 7B and 14B parameter models with high throughput, the reasoning performance of the local models frequently lagged behind the state-of-the-art frontier models. I found that the additional cognitive overhead of correcting errors in local model output outweighed the benefits of offline execution, leading me to rely primarily on the cloud-hosted agents for complex implementation tasks.

D.3 Division of Responsibility

My working rule throughout the project was for me to handle the engineering and for the agents to flesh out the implementation. In practice, this meant that I was responsible for every decision that had downstream consequences for the shape of the system, and the agents were responsible for producing code that realized those

decisions. Concretely, I did the following work directly, without delegating it to an agent:

- **Architecture.** The three-tier structure described in Chapter 4, the separation between experiment specifications and trial records, the choice to route all robot communication through plugin files, and the overall shape of the event-driven execution model were mine. I wrote these decisions as prose before any code was written.
- **Data model.** The PostgreSQL schema and the tRPC procedure boundaries were designed by me. Because downstream type safety depends on the shape of the schema and the API, I was unwilling to let an agent make those choices.
- **Research design.** The pilot validation study in Chapter 6 was designed and analyzed entirely by me. The Observer Data Sheet, Design Fidelity Score rubric, and Execution Reliability Score rubric were written by hand. No AI tool was used to score sessions, compute results, or draft claims about what the data showed.
- **The prose of this thesis.** Every chapter was written by me. The structure of the argument and the specific claims I make are my own. While AI assisted with the nuances of LaTeX formatting (particularly the generation of TikZ diagrams and complex chart syntax), the content is mine.

D.4 Evolution of the Workflow

My use of these tools changed over the course of the project, and evolved as the models improved. Early on, I treated the agent's output as a draft that required line-by-line review. The typical loop followed five steps: writing a specification, generating a diff, reading the diff, running the code, and then accepting or rejecting the change.

As the models improved and the agents became more reliable, the focus of my effort shifted. By the final stages of development, I spent significantly less time on manual line-by-line reviews and more time on empirical testing. I moved from being a "code reviewer" to a "test-driven supervisor." If the agent produced a feature that passed my manual acceptance tests and integrated correctly with the existing system, I was more likely to accept the implementation without auditing every line in the program. This shift allowed me to increase the velocity of development significantly in the weeks leading up to the evaluation.

D.5 What Worked and What Did Not

The tasks that agents handled well were those with a narrow and well-specified interface. Implementing a tRPC procedure from a signature, writing a Drizzle migration that matched a schema diff, adding a new field through an existing form, or applying a consistent rename across files: these were cheap to specify and the agent's output was usually accepted on the first or second iteration. Agents were also good at scaffolding: producing the initial shape of a component, test file, or API route that I then edited to completion.

The tasks that agents handled poorly were those that required reasoning across more of the system than the context window could hold, or that depended on a piece of context I had not written down. Cross-cutting changes to the experiment and trial data models, for example, required careful coordination across the schema, the tRPC procedures, the execution runtime, and the analysis interface. When I tried to delegate changes of this shape to an agent, the diffs were often locally plausible but globally inconsistent; I ended up doing that work myself. Subtle concurrency and timing questions in the execution layer were another category the agents did not handle well. The event-driven execution model in Chapter 4 has enough non-obvious ordering constraints that an agent without the full picture tended to introduce races; those parts of the codebase I wrote by hand.

Across the full set of tools I used, the differences in capability for the work I asked of them were smaller than I expected. Any of the agents could, in principle, produce a correct diff for a well-scoped task, and when one tool failed it was usually because the task was underspecified rather than because of a difference in model capability. The practical differences between tools mattered more at the workflow level, such as which shell integration I preferred, how the tool handled long diffs, and how it behaved when it needed to ask for clarification, than at the capability level.

D.6 Research Integrity

Because this thesis reports an empirical evaluation, I treat the boundary between AI-assisted development and the evaluation itself as a matter of research integrity rather than a matter of preference. The following statements reflect the actual workflow I

followed:

- No AI tool generated, modified, or interpreted any of the evaluation data reported in Chapter 7. Every Design Fidelity Score, Execution Reliability Score, and System Usability Scale rating was recorded by me during or immediately after each session from direct observation, using the rubrics in Appendix A.

- No AI tool produced the tables, means, or comparative claims in Chapter 7. The numbers were tabulated by hand from the completed Observer Data Sheets reproduced in Appendix B, and the claims about what those numbers support or do not support are mine.

- No AI tool drafted the prose of this thesis. The chapters were written by me, in my own voice, and I am responsible for every claim they make and every argument they advance. AI tools were occasionally used as a proofreading aid to catch typos, flag awkward phrasing, or suggest an alternative word; however, the sentences are mine.

- The code that implements HRIStudio and that was the subject of the evaluation was written under the workflow described in Sections D.3 and D.4. Agents produced drafts; I read, tested, and accepted or rejected every one. The final state of the code is the product of my engineering decisions, regardless of who wrote any particular line.

D.7 A Note on the Workflow as a Contribution

The workflow described in this appendix is not a contribution of the thesis, and I do not claim that it is generalizable or optimal. I describe it because it is the actual workflow under which the system was built, and because a reader evaluating the claims in Chapter 7 is entitled to know how the system being evaluated came into existence.

The more interesting observation, at least to me, is about where the boundary between human and agent naturally fell in practice. It fell at the point where a task required a decision with downstream consequences for the shape of the system. Tasks that realized a decision were inexpensive to delegate and inexpensive to verify; tasks that made a decision were neither, and delegating them produced diffs that were locally plausible and globally wrong. Whether that boundary will move as tools improve is a question I cannot answer from the evidence of a single project, but the boundary was stable across every tool I used during this one.