TU WIEN Informatics

Requirements-Driven Tests for Cognitive Architectures

AERA as a Case Study

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering

eingereicht von

Maximilian Pfeil, BSc
Matrikelnummer 11809919

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.-Prof. Dr. sc. Jürgen Cito
Zweitbetreuung: Prof. Dr. Kristinn R. Thórisson

Wien, 12. August 2026

Maximilian Pfeil	Jürgen Cito

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Maximilian Pfeil, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, haben ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT- Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 12. August 2026

Maximilian Pfeil

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acknowledgements

I would like to thank Prof. Jürgen Cito for offering to supervise a thesis on a topic I proposed myself, and for his guidance throughout this work.

My thanks go to Prof. Kristinn R. Thórisson for introducing me to the research field of cognitive architectures, as well as for his support throughout the entire course of this thesis.

I want to thank Prof. Olivier Georgeon and Prof. Sean Kugele for taking the time to answer several of my early questions and for helping to shape my research direction.

Furthermore, my thanks go to the AERA development and research team, especially Jeff Thompson, Leonard Eberding, and Chloe Schaff, for their technical support and topical discussions.

I am also grateful to my parents, Edith and Klaus, and my sisters, Isabella and Victoria, for their support throughout my studies at TU Wien.

Finally, I thank Laura for her continuous encouragement and inspiration while discussing major philosophical questions as well as minor formal details with me.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Kurzfassung

Vom Turing-Test bis hin zu modernen Benchmarks wie ARC-AGI ist aufgabenbasiertes Testen ein zentrales Mittel zur Evaluation von KI-Systemen. Moderne KI-Systeme können Aufgaben lösen, die für Menschen allgemeine Intelligenz erfordern würden, zeigen aber selbst keine allgemeine Intelligenz. Kognitive Architekturen werden erforscht, um die Lücke zwischen statistischen Methoden und natürlicher Intelligenz zu schließen. Die Intelligenzanforderungen an solche Architekturen sind jedoch oft vage definiert und werden beim Testen von Softwareimplementierungen solcher Systeme nicht berücksichtigt. Bestehende Aufgabenumgebungen sind typischerweise ad hoc entworfen, was eine systematische Bewertung und einen Vergleich verschiedener Architekturen erschwert. Wir präsentieren eine Methode, die die Intelligenzanforderungen einer kognitiven Architektur als Softwarespezifikation behandelt und diese Anforderungen spezifischen Architekturmechanismen zuordnet, um konkrete Testaufgaben abzuleiten. Wir haben diesen Ansatz auf die Autonomous Empirical Reasoning Architecture (AERA) angewendet und 20 ihrer 32 Anforderungen in 57 Aufgabenumgebungen getestet. Die resultierende Testsuite deckt große Teile der AERA-Dokumentation ab und unterstützt Evaluierung, Entwicklung und Lehre. Unser Ansatz ist auf jede kognitive Architektur mit explizit spezifizierten Intelligenzanforderungen und einer White-Box-Softwareimplementierung erweiterbar.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

From the Turing Test to modern benchmarks such as ARC-AGI, task-based testing has remained central to the software evaluation of AI systems. Contemporary AI systems can solve tasks that would require general intelligence if done by humans, yet do not exhibit general intelligence themselves. Cognitive architectures have been pursued by researchers to bridge the gap between statistical methods and natural intelligence. However, intelligence requirements for such architectures are often loosely defined and not taken into account when testing software implementations of such systems. Existing task-environments are typically designed ad hoc, making principled evaluation and comparison across architectures difficult. We present a method that treats a cognitive architecture's intelligence requirements as a software specification, mapping these requirements to specific architectural mechanisms to derive concrete test tasks. We have applied this approach to the Autonomous Empirical Reasoning Architecture (AERA), testing 20 of its 32 requirements across 57 unit task-environments. The resulting test suite covers large parts of AERA's documentation and supports evaluation, development, and education. Our approach is extensible to any cognitive architecture with explicitly specified intelligence requirements and a white-box software implementation.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Kurzfassung **vii**

Abstract **ix**

Contents **xi**

1 Introduction **1**
1.1 Problem statement . . . 2
1.2 Research questions & hypotheses . . . 3
1.3 Summary of contributions . . . 3

2 Theoretical Foundations **5**
2.1 Cognitive architectures . . . 5
2.2 AERA . . . 6
2.3 Replicode . . . 7
2.4 Specification-based and requirements-based testing . . . 9

3 Related Work **11**
3.1 Testing AI systems . . . 11
3.2 Benchmark environments . . . 13
3.3 Ability-oriented testing . . . 14
3.4 Task theory . . . 15
3.5 Testing in practice . . . 16
3.6 Answering RQ1 . . . 17

4 Methodology **19**
4.1 Preliminary testing . . . 19
4.2 Intelligence requirement table . . . 20
4.3 Test design . . . 21
4.4 Evaluation against related frameworks . . . 24
4.5 Threats to validity . . . 24

5 Results **25**
5.1 Test artifacts . . . 25

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

5.2 Strengths of the implementation 26
5.3 Limitations of the implementation 28
5.4 Reviewing requirements for ability-oriented testing 29
5.5 Reviewing requirements for a task theory 29
5.6 Answering RQ2 31

6 Discussion **33**
6.1 Requirement-driven tests for cognitive architectures 33
6.2 Generalizability to other cognitive architectures 34
6.3 Answering RQ3 35

7 Conclusion **37**
7.1 Key contributions 37
7.2 Future work 38

A Intelligence Requirement Table **39**

B Test Documentation **43**

Overview of Generative AI Tools Used **49**

List of Figures **51**

List of Tables **53**

Bibliography **55**

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 1

Introduction

Verifying that a software system behaves as intended is a fundamental concern in software engineering. For AI systems, task-based evaluation has a long history in this regard, from the Turing Test [32] to modern benchmarks such as ARC-AGI [3]. However, task performance alone is an insufficient criterion. AI systems can solve tasks that would require general intelligence if performed by humans, without actually implementing the cognitive mechanisms those tasks are intended to measure [10]. Therefore, it is not sufficient to test whether a system produces correct outputs, but necessary to verify that the underlying cognitive mechanisms operate as specified.

Cognitive architectures such as Soar [16] and NARS [34] represent a research direction that addresses this by grounding system design in explicit theories of intelligence rather than optimizing for performance on predefined tasks [13]. Because these architectures define explicit cognitive requirements, their implementations are in principle verifiable against a specification. In practice, however, these requirements are often loosely specified and rarely reflected in how implementations are tested. Evaluation tends to rely on general-purpose benchmarks that abstract over architecture-specific properties, or on demonstrations that showcase many cognitive abilities at once. Task-environments for system testing are typically constructed ad hoc, making systematic evaluation and comparison across architectures difficult.

Our work does not propose a new evaluation benchmark for AGI-aspiring architectures. Instead, we explore the approach of treating a cognitive architecture's intelligence requirements as a software specification. We use this specification to systematically derive concrete test cases, similar to how requirements-based testing is practiced in traditional software engineering [12]. We apply this approach to the **Autonomous Empirical Reasoning Architecture (AERA)**, which is under development at the Center for Analysis & Design of Intelligent Agents (CADIA) at Reykjavik University and the Icelandic Institute for Intelligent Machines (IIIM). This yields a test suite in which each test case is traceable to a specific intelligence requirement and the architectural mechanism

by which it is realized. While we do not claim to fulfill a formalized task theory, we document our test artifacts in a way that takes a step towards one.

This thesis is structured as follows: The remainder of Chapter 1 presents the problem statement, our research questions and hypotheses, and a summary of our contributions. Chapter 2 describes the theoretical foundations of our work, including the AERA system. Chapter 3 reviews related work on how cognitive architectures are tested. Chapter 4 describes our methodology, i.e., how we derived and designed tests for AERA. Chapter 5 presents the results of our test process. Chapter 6 discusses our findings in a more general context. Chapter 7 concludes the thesis.

1.1 Problem statement

While general-purpose benchmarks to measure the performance of AI agents exist, their metrics may not account for the specific intelligence requirements of the system under test. For example, François Chollet, in his proposed ARC-AGI benchmarks, focuses primarily on generalizability [3], which does not cover all aspects of intelligence required by AERA's design principles, such as cumulative learning or constructivist design. Comparing different task-environments, benchmarks, and agent performance in a meaningful way is further complicated by the absence of an established task theory in AI [30].

Furthermore, cognitive architectures are often the work of small development teams, and their implementations do not provide the guarantees expected of industrial-strength software. Many remain incomplete in parts, and often lack a systematic test suite [14]. Existing evaluations primarily take the form of demonstrations[1,2] that aim to show a broad range of intelligence-related features simultaneously. While informative, such demonstrations do not establish a clear mapping from individual intelligence requirements to corresponding tests, making it difficult to assess which specific cognitive properties are satisfied and which are not.

With regard to AERA, the system's cognitive processes are implemented in Replicode, a programming language which had to be specifically created to satisfy the system's intelligence requirements [20]. AERA's autonomic, constructivist, and non-axiomatic design methodology is described in publications that specify a set of intelligence requirements its implementation is intended to satisfy. As an explainable white-box system supporting reflection and inspection, it provides the transparency necessary to evaluate internal cognitive mechanisms directly. This is a prerequisite for the requirements-driven testing approach proposed here. Building on this motivation, we now turn to the research questions and hypotheses guiding this thesis.

[1] OpenAERA demos, accessed 12 August 2026.
[2] OpenNARS demos, accessed 12 August 2026.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

1.2 Research questions & hypotheses

RQ1 What test strategies do existing cognitive architectures follow?

H1 Just as different cognitive architectures have distinct approaches to understanding the phenomenon of intelligence, they also adopt very different strategies for ensuring that their implementations meet the intended requirements.

RQ2 Which of AERA's special properties can be tested in small, requirements-driven task-environments?

H2 Like other cognitive architectures, AERA specifies distinct requirements for intelligence (e.g., life-long learning, defeasible reasoning). From this requirement list, small task-environments can be derived, implemented as tests, and used to evaluate specific cognitive processes; similar to how traditional software tests are derived from a specification.

RQ3 To what extent can the testing approach developed for AERA in RQ2 be generalized to other AGI-aspiring systems?

H3 The test creation approach described in H2 can be generalized by substituting AERA's requirement list with that of other cognitive architectures.

1.3 Summary of contributions

This thesis makes several contributions to the integration of software tests in AGI-aspiring systems. We first explore and organize the software-theoretical foundations for testing cognitive architectures. Then, we provide an exploratory overview of existing evaluation methods and testing strategies for such systems. Building on these insights, we take a requirements-driven approach to derive, implement, and evaluate small task-environments for AERA. The resulting test artifacts are documented in a way that takes a step toward a more principled task theory, without claiming to fulfill one. Finally, we discuss the potential generalization of this approach to other cognitive architectures.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 2

Theoretical Foundations

In this chapter, we present the theoretical background necessary for our work. We outline the landscape of cognitive architectures and provide a brief overview of selected examples. We also explain our choice of AERA for the case study, and describe the system in more detail. Furthermore, we briefly discuss the motivation for and methodology behind specification-based and requirements-based testing.

2.1 Cognitive architectures

Scientific interest in general artificial intelligence began in the 1950s, with cognitive architectures emerging as one of its research directions [13]. The overarching goal of the field is to develop systems with human-like intelligence. How exactly such systems should operate remains a subject of debate, much like the phenomenon of intelligence itself [17]. Cognitive architectures differ from traditional software through additional requirements such as meta-cognition, i.e., the ability to reflect on and adapt one's behavior across a variety of tasks including unforeseen situations [34]. Further requirements include physical embodiment and operation under insufficient knowledge or resources, such as time and resource constraints. The implications of these additional requirements span all stages of development, from design to testing.

Various methodologies have been proposed to build such systems. In 2003, Russell and Norvig describe cognitive architectures as models of human reasoning and highlight the popularity of symbolic production systems in the field [24]. In a 2020 survey, Kotseruba and Tsotsos review 84 cognitive architectures, 49 of which were still actively developed at the time [13], categorizing them into symbolic, emergent, and hybrid approaches, each with distinct strengths and weaknesses. Emergent approaches, for instance, excel at massively parallel modeling but sacrifice transparency and explainability. To illustrate the landscape of cognitive architectures and their differing key requirements, we briefly present three examples; AERA itself is introduced in Section 2.2.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Soar is one of the oldest and longest-running cognitive architectures [16]. Originally created in the early 1980s by John Laird, it provides fixed computational building blocks for creating AI agents [15]. While initially targeting multi-task, multi-method problem solving, the system has evolved to support reinforcement learning, episodic and semantic memory, and a visual system.

LIDA (Learning Intelligent Decision Agent) is another established cognitive architecture [13], described by its creator Stan Franklin as "a systems-level architecture for cognition, emotion, and learning" [6]. Largely based on the Global Workspace Theory by Baars [1], it has since evolved to include a variety of cognitive modules and processes for attention, action selection, and learning.

NARS (Non-Axiomatic Reasoning System) is designed specifically to operate under the assumption of insufficient knowledge and resources proposed by Pei Wang [33]. This involves dealing with finite processing power and storage space, operating in real time, and handling tasks of general content. These features allow the system to function in the physical world, consistent with Wang's definition of intelligence.

From a software architecture perspective, it is important to distinguish cognitive architectures from their concrete software implementations [13]. For example, several implementations of Soar demonstrate only a subset of the full architecture's capabilities, but may also introduce features not present in the original design.

For our work, we required a cognitive architecture that satisfies several criteria: (1) it must have a software implementation against which tests can be written; (2) the implementation must be sufficiently documented to allow verification of test results; (3) the architecture must follow an explainable design, allowing for tests that inspect internal processes; and (4) the underlying methodology must be well described in the literature, enabling us to derive a list of requirements and, subsequently, corresponding test cases.

AERA has an open-source implementation[1], documentation in the form of a technical report [20], and follows a fundamentally explainable design methodology described in numerous research papers [21, 26, 28]. Based on these criteria, we chose AERA as the cognitive architecture for our case study. The next section describes the system in more detail.

2.2 AERA

In Chapter 1, we briefly introduced the Autonomous Empirical Reasoning Architecture (AERA). In this section, we describe the system's history, principles, goals, and inner workings in more detail.

AERA was originally designed by Eric Nivel and Kristinn R. Thórisson in collaboration with researchers across the EU as part of the HUMANOBS project [21]. A prototype of

[1] IIIM-IS/AERA on GitHub, accessed 12 August 2026.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

the architecture, originally referred to as the Autocatalytic Endogenous Reflective Architecture, was implemented. Within the HUMANOBS project, it was used to demonstrate the feasibility of taking a constructivist approach to creating a system capable of learning complex tasks and skills through observation [21]. This was demonstrated in a scenario that replicated a one-on-one TV-style interview of real-time multimodal dialogue. The architecture has continued to develop since, incorporating new features. For example, its internal handling of time has been modified, and most recently a mechanism for reasoning by analogy has been introduced [25].

Collecting AERA's design principles will be central in Chapter 4, where we derive a test specification from the system's intelligence requirements. As a preview, the following three high-level design goals are key elements in AERA's design: (1) to operate effectively in the real world, an intelligent agent must be able to interact with its environment through sensing and acting, as such interaction is necessary to acquire knowledge of causal relationships [21]; (2) the system must construct its knowledge autonomously in a constructivist manner [26]; and (3) like NARS, AERA is designed to operate under the Assumption of Insufficient Knowledge and Resources (AIKR) [33, 21]. This implies that time and resource management are transversal concerns and that the system must support non-axiomatic reasoning.

We illustrate these core principles by describing how AERA handles a simple task. In line with its constructivist design, the agent is initialized with a small seed from which it builds its knowledge. This seed contains small causal models of the real world, such as the knowledge that when the faucet valve on a bathtub is opened, the water level will gradually increase. When the agent is given the goal of filling the bathtub to a certain level, it first uses its sensors to detect the current state of the world. It then derives a plan from this state to the desired goal state using its models, while respecting time and resource constraints. Finally, the agent uses its actuators to open the faucet and observes whether its predictions about the real world hold. Since any knowledge may become incorrect at any time, consistent with the AIKR, the strength of existing models is updated autonomously. Models that perform poorly over time are eventually removed from the system [21].

This overview only gave a concise introduction to the functionality of AERA. More detailed mechanisms will be revisited in later chapters. The next section describes AERA's programming language, Replicode.

2.3 Replicode

Replicode, an interpreted language implemented in C++, specifies knowledge and algorithms capable of reasoning about knowledge and algorithms in a parallelized manner [20]. The language had to be specifically created to satisfy the unified requirements of AERA, which no other programming language fulfilled[2]. Replicode is the work of a small devel-

[2]OpenAERA website, accessed 12 August 2026.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

opment team, and therefore does not come with the kinds of guarantees expected from industrial-strength programming languages. Our test cases are written exclusively in Replicode.

The following are key characteristics of the language as described in its technical documentation [20]:

Pattern matching: Knowledge in Replicode is represented as objects (e.g., models, facts, predictions), and any such knowledge can serve as the target of a pattern.

Dynamic code production: Using pattern matching, Replicode injects knowledge into its many parallelizable template models and programs, producing instantiated models and programs.

Reflection: To support introspection, the Replicode executive injects knowledge of the system's internal processes, which can then be used again for pattern matching.

Model-driven: Replicode models, once instantiated, output productions, which can be used in forward chaining to create predictions, or in backward chaining to create sub-goals.

Non-axiomatic: Knowledge does not represent absolute truth, but rather an incomplete, context-dependent reflection of the system's experience up to that point.

Self-programming: Learning is accomplished through pattern extraction. Replicode induces new models and states from changes in the environment, failed predictions and goal achievement.

To illustrate Replicode's design and syntax, we break down the structure of the model shown in Listing 2.1. The model `M_WAIT` encodes the causal knowledge that when the agent waits in the presence of a bathtub with an open faucet valve, the water level will increase. This knowledge can be used to generate either a prediction ("What will happen if I wait while the water is running?") or a sub-goal ("What do I need to do to cause the water level to rise?"). The corresponding requirement model, which adds the constraint that `M_WAIT` is only applicable if the bathtub's faucet valve is open, is omitted for brevity.

In line 2, the model `M_WAIT` is declared using the keyword `mdl`, along with a set of template arguments. These arguments are instantiated with actual values once matching patterns are found. Line 3 specifies the left-hand side of the model: the agent executes the command `wait`. Line 4 specifies the right-hand side: the bathtub's water level is `WL1`. Lines 5–9 define the forward guards of the model, which calculate the change from `WL0` to `WL1` and enforce time and state constraints; these are used when generating predictions (left-to-right). Lines 11–16 define the backward guards, which are applied when generating sub-goals (right-to-left). Finally, line 17 specifies control values of the model (e.g., execution count and success rate) and the model's view [20].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Listing 2.1: A causal model in Replicode

```
; When I wait in the presence of a bathtub, the water level will rise
M_WAIT:(mdl [B: WL0: (ti T0: T1:)] []
    (fact (cmd wait [B:] :) T0_cmd: T1_cmd: : :)
    (fact (mk.val B: water_level WL1: :) T0_rhs: T1_rhs: : :)
[]
    (< WL0 WL_FULL)
    WL1:(+ WL0 WL_DELTA)
    T0_rhs:(+ T0 100ms)
    T1_rhs:(+ T1 100ms)
[]
    (> WL1 WL_EMPTY)
    WL0:(- WL1 WL_DELTA)
    T0_cmd:(- T0_rhs 80ms)
    T1_cmd:(- T1_rhs 100ms)
    T0:(- T0_rhs 100ms)
    T1:(- T1_rhs 100ms)
[stdin] 1 53 0.95 0.95 1) [[SYNC_ONCE now 0 forever primary nil 1]]
```

For our tests, we will write numerous tasks composed of similar models, programs, and states, each focusing on different characteristics of the Replicode language. The final section of this chapter provides an overview of relevant approaches to software testing.

2.4 Specification-based and requirements-based testing

Specification-based testing, also referred to as black-box testing, is a testing strategy in which "the principal test basis is the external inputs and outputs of the test item, commonly based on a specification, rather than its implementation in source code or executable software" [12]. Instead of deriving tests from the source code, test cases are constructed from input-output relations defined by a specification of the system's expected behavior. Consequently, the effectiveness of this approach depends on the completeness and accuracy of the underlying specification; if the specification is incomplete or outdated, test quality will suffer accordingly [12].

Requirements-based testing is a specific form of specification-based testing in which the test cases are derived directly from the requirements of the system under test. Its goal is to verify that the system satisfies the needs of its intended users [12]. Like in specification-based testing, the usefulness of this approach depends heavily on the quality of the requirement list. If requirements are incomplete or poorly specified, the resulting tests will provide limited coverage.

In our work, we combine both approaches. From a requirement list for intelligence and a specification in the form of the Replicode technical report [20], we derive test cases for the AERA system. This process is described in Chapter 4.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 3

Related Work

In this chapter, we discuss existing testing methods for AI systems and analyze how they relate to our approach. The first section establishes the landscape and identifies limitations of current testing approaches. The second section presents benchmark environments such as Gymnasium and ARC-AGI. The third section discusses the difference between task-oriented and ability-oriented testing, and the fourth section task theory. The fifth section reviews how testing is done in practice, by examining the code repositories of two cognitive architecture implementations. The final section draws on the preceding analysis to answer our first research question.

3.1 Testing AI systems

Before discussing *how* AI systems can be tested, we briefly describe *why* and *what* needs to be tested, in accordance with Bieger et al. [2]. For general AI systems, reasons to test include: (1) to assess task-specific performance, (2) to gain knowledge of the system's strengths and weaknesses, and (3) to ensure predictable and trustworthy behavior [2]. Regarding what to test, certain properties are inherent to the system's design, such as the specifics of a planning algorithm, and can be verified by inspecting the implementation directly. Others are better assessed empirically through tasks, such as the amount of prior knowledge required to acquire a new skill [2]. We now shift our focus towards how to test.

In 2007, Legg and Hutter survey the most prominent proposed tests for artificial intelligence [18]. They begin with Alan Turing's imitation game, which holds that a machine should be considered intelligent if human observers cannot distinguish it from a human during text-based communication [32]. While acknowledging the test's simplicity, Legg and Hutter argue that it is neither necessary nor sufficient as a measure of intelligence: not necessary, because it tests humanness rather than intelligence, and not sufficient, because a system may merely appear intelligent without actually possessing intelligence [18]. The

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

latter point aligns with Hernández-Orallo's 2017 observation that "AI artifacts solve tasks that would require intelligence if done by humans without featuring intelligence" [10]. This supports the notion that meaningful evaluation of AI systems requires a deeper understanding of the components that compose intelligence, perhaps through tests that target these components directly. This supports our requirement-driven approach, and is discussed more in Section 3.3.

Besides the Turing test and its derivatives, Legg and Hutter examine seven other testing approaches, evaluating each against properties such as validity, generality, and practicality. Their comparative overview concludes that no single approach, including their own "Universal Intelligence" test, satisfies all desired properties, and therefore no single evaluation method for machine intelligence exists [18]. In 2010, Hernández-Orallo and Dowe build upon Legg and Hutter's test, presenting the concept of a universal anytime intelligence test [11]. Here, "universal" means applicable to any intelligent system, biological or computational, and "anytime" means the test can be conducted at any time scale and interrupted at any point. However, they later acknowledged practical limitations when the approach was implemented [10]. Section 3.4 discusses how a principled task theory could enable practical tests while still covering a large number of test cases.

As described by Olivier Georgeon [7], all possible tests for AI systems can be classified into four quadrants, which are shown in Table 3.1. The first dimension distinguishes between one-shot and continuous interaction tests. In a one-shot test, such as in traditional unit tests or when prompting a Large Language Model (LLM), the system under test is not expected to learn or improve during the test session. By contrast, reinforcement learning is an example of a continuous interaction test, where the agent iteratively learns from feedback provided by the test environment. The second dimension differentiates between quantitative and qualitative evaluation. Quantitative measures yield a scalar performance value, as in Massive Multitask Language Understanding (MMLU) [9], whereas qualitative evaluation relies on human judgment, as in the Turing test, or when a human inspects answers of an LLM or log traces of an explainable cognitive architecture. It should be noted that the quadrant a test falls into is not solely determined by the test itself, but also by the system under test. A system that learns during the Turing test would shift it from a one-shot to a continuous interaction test. As part of our work, we developed tests fitting all four quadrants of Table 3.1: one-shot unit tests check basic Replicode syntax, and AERA's intelligence features are evaluated in more complex tasks through continuous interaction with the environment. Both one-shot and continuous interaction tests can be analyzed in a quantitative or qualitative manner, depending on whether the test result alone is examined or a developer inspects the test case more deeply.

	One-shot	**Continuous interaction**
Qualitative	LLM prompts (human judge)	Demos for AERA or ECA
Quantitative	Unit tests, MMLU	Reinforcement learning

Table 3.1: Four categories of artificial intelligence tests with examples.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

As part of his own work, Georgeon introduced the Enactivist Cognitive Architecture (ECA), grounded in the theory of enaction [8]. ECA is evaluated in multiple ways. Traditional unit tests verify that individual implemented functions behave as expected, as in any software development project. Testing for genuine features of intelligence, such as learning, is achieved through two complementary approaches: inspection of logs and traces to observe the agent's internal processes, and demonstrations to human judges, both of which fall into the upper quadrants of Table 3.1. Georgeon argues that because AI systems address different aspects of intelligence, demonstrations tend to be designed specifically for each system, and different research tracks do not converge on a shared testbed; a gap the field still needs to address [7].

This gap is also acknowledged by Sean Kugele [14], a researcher working on LIDA, one of the cognitive architectures introduced in Section 2.1. As of 2025, LIDA's development is somewhat federated, with different teams building different agents and a shared evaluation framework still a work in progress. Kugele proposes shared task-environments, explorable by both human and AI agents, where the expectation is that behavior should be comparable across both [14]. Meaningful comparison would require a principled task theory, which is the topic of Section 3.4. First, however, we describe benchmark environments, the current approach to comparing AI agent performance, and their limitations.

3.2 Benchmark environments

With common input/output structures, tasks should be applicable to many different AI agents, resulting in benchmark tests that, while having to abstract over architecture-specific workings, would at least provide some basis for comparing them [14]. Notable examples of such frameworks are Gymnasium and ARC-AGI.

Gymnasium, the maintained successor of OpenAI Gym, is an open-source library that provides a standardized API for reinforcement learning (RL) task-environments [31]. In addition to easy-to-use environments, Gymnasium offers various tools to assist RL research. However, since it only supports RL agents, most cognitive architectures cannot be evaluated using its environments.

The Abstraction and Reasoning Corpus (ARC, or ARC-AGI) was introduced in 2019 by François Chollet [3]. It has since become one of the most prominent benchmarks for evaluating progress toward AGI, with 1,455 teams and 15,154 entries participating in its most recent iteration in 2025, including frontier AI labs such as Anthropic, Google DeepMind, OpenAI, and xAI [5]. A core design principle of ARC-AGI is that its tasks should be easily solvable by humans yet challenging for AI systems. Chollet's motivation for the tests was that while AI systems had become highly skilled at specific tasks, this was accomplished in a fundamentally non-human way, by relying on vast amounts of training data and priors [3]. This resulted in agents that struggle to adapt to unforeseen tasks. It is precisely this ability to generalize and adapt that ARC-AGI aims to test. Thus, ARC-AGI tests for aspects of intelligence that are central to many cognitive architectures,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

such as generality and adaptation. However, other requirements integral to some cognitive architectures, such as explainability and the ability to adapt under the assumption of insufficient knowledge or resources, remain outside the scope of ARC-AGI-2 [5].

According to Chollet, ARC-AGI is not an indicator of whether AGI has been achieved, but rather a compass pointing toward meaningful research directions and a tool to accelerate progress [4]. The next two sections present proposals on how evaluation could be further improved: by shifting from task-oriented to ability-oriented evaluation, and by following a principled task theory.

3.3 Ability-oriented testing

In 2017, Hernández-Orallo describes differences in task-oriented and ability-oriented measurements when evaluating AI [10]. He differentiates between AI applications and AI systems: while task-oriented evaluation works well for narrow AI, which is designed to excel at a specific task, this does not suit AGI-aspiring systems. For example, in order to cover the wide range of unforeseen tasks a cognitive robot should be able to do, it needs a variety of skills like learning and reasoning. Whereas tasks are measuring instruments, these properties are constructs that need to be conceptualized before they can be evaluated [10]. Hernández-Orallo argues that the goal of AGI testing should not be to evaluate systems for what they do, but for what they are able to learn to do.

Some benchmarks similar to those we discussed in Section 3.2 appear ability-oriented on the surface, but Hernández-Orallo argues this is not the case. For example, the International Planning Competition appears to test the skill to plan rather than a single task. However, participating systems are built with planning as their goal, not a resource: planning is their hard-coded purpose rather than a capability they deploy when needed [10]. Another concern is that instances of evaluation overfitting are not properly analyzed by benchmark designers. Rather than addressing this directly, the problem is often circumvented by the addition of new tasks or minor modifications to the evaluation process. Furthermore, systems that performed well at benchmark competitions often showcased brittle learning, where small changes in the task-environment led to the system having to relearn everything from scratch. Ability-oriented approaches from psychometrics do not suffice either, as they tend to follow an anthropocentric design. This limitation became apparent when attempts to generalize IQ tests from humans to the animal world were deemed “chimpocentric” [10].

Another consideration in ability-oriented evaluation is to look beyond the final results of a task. Many evaluation methods deliver only a single aggregate value, discarding much of the behavioral data collected during the test [2]. More broadly, the interesting question is not whether a goal was achieved, but how the agent’s internal processes went about achieving it. While a simple measure of success might be a score in a video game, or the final answers in a multiple choice test, the deeper insight lies in the agent’s underlying behavior. Ultimately, any task, including the Turing test, can be reduced to one or more yes-or-no questions regarding timing effects or intermediate behavioral milestones [14].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Where black-box evaluation is desirable for fair, objective comparison between systems, whether biological or artificial, white-box evaluation is especially useful for debugging, identifying weak points, or assessing specific cognitive abilities [2].

In order to follow a more ability-oriented approach, Hernández-Orallo concludes with the following guidelines for future benchmark competitions: (1) it should be specified beforehand which systems, be it humans, fine-tuned AI or autonomous AI, are to be evaluated, (2) the set of possible tasks and their associated distribution should be defined, (3) the scoring function must produce comparable, bounded values across all tasks, (4) similarities between tasks should be identified, (5) the sampling method should be as efficient as possible, (6) the information disclosed to participants should be carefully chosen, and (7) evaluation results should be analyzed beyond the simple aggregation of scores [10]. We will discuss to what extent our tests fulfill these requirements in Chapter 5.

Of these guidelines, points 2-4 and 7 all relate to the design and comparison of tasks. Meaningful analysis of AI system performance across different task-environments is severely limited without a principled task theory, the topic of the next section.

3.4 Task theory

In 2016, Thórisson et al. argue that the ad hoc design of tasks and evaluation environments is a fundamental weakness of current AI evaluation, and propose a task theory to make the construction, adaptation, and comparison of task-environments more principled [30]. While procedural task generation has been proposed, the associated near-complete randomness makes it difficult to relate to real-world scenarios [19]. As such, a task theory does not need to represent every possible task, but only a sufficiently wide range. Establishing a task theory could support AGI research in three key aspects: evaluation, pedagogy, and design:

Evaluation. While specific requirements for intelligence remain debated, this does not prevent a task theory from being established. We can measure intermediate milestones towards cognitive abilities, despite different philosophies. We do not need all systems to run on the same test set, if we can compare tasks. This would allow each system to act in its "natural habitat" while still being comparable to others [30].

Pedagogy. A task theory that allows us to scale task difficulty and size, for example by decomposing it into sub-tasks, would allow us to teach AI systems easier versions before tackling a more difficult version of the same task [30].

Design. A task theory would help alleviate the trial and error involved in designing AI systems for a particular task [30].

As for its requirements, Thórisson et al. argue that a task theory should support the following: (1) task comparison, (2) task abstraction and concretization, (3) estimation of time and resource requirements for task completion, (4) characterization of task

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

complexity, (5) easy task construction through composition, decomposition and scaling, (6) procedural generation, and (7) transparency to analysis [30, 29]. They further concretize how key concepts could be formalized in a task theory, including the environment, its state, the agent, problems, goals, and tasks with time and resource requirements.

Task-environments created as part of a task theory should support the following: (1) both determinism and partial stochasticity, (2) ergodicity, i.e., different degrees of second chances an agent gets, (3) controllable continuity through continuous variables, (4) asynchronicity across all sensors and controls, (5) dynamic or static state handling, (6) different degrees of partial observability, (7) different degrees of controllability, (8) multiple parallel causal chains, (9) varying number of agents, (10) periodicity, and (11) repeatability [29].

The importance of a task theory has been central throughout this chapter. We will therefore use these specifications as a reference point when designing and documenting our test artifacts, and review them when analyzing which requirements are fulfilled by the task-environments implemented as part of this work, without claiming to fulfill a formalized task theory. The next section in this chapter examines the code repositories of two cognitive architecture implementations to assess what testing strategies are followed in practice.

3.5 Testing in practice

Before concluding this chapter, we briefly discuss how the theoretical concepts mentioned so far are applied in practice. The following is based on inspection of the code repositories and websites of two implementations of cognitive architectures: ONA[1] (OpenNARS-for-Applications) and AERA[2,3].

Both repositories contain tests spanning all four quadrants of Table 3.1. One-shot quantitative tests include unit tests targeting individual software components. Additionally, both repositories contain system tests in the form of task-environments, which can either be run automatically to yield a scalar performance score, or used by a human reviewer to inspect log traces of internal processes. Several video demonstrations and teaching materials reference specific task-environments. In both repositories, we found task-environments that had been adapted from simpler variants, requiring the system to employ cognitive abilities not previously needed. Systematically scaling such task-environments would benefit greatly from the aforementioned task theory. Test specification and documentation are largely absent; understanding the purpose or targeted ability of most test cases requires familiarity with the system's design and its representation language (e.g., Replicode).

[1] OpenNARS for Applications on GitHub, accessed 12 August 2026.
[2] IIIM-IS/AERA on GitHub, accessed 12 August 2026.
[3] OpenAERA website, accessed 12 August 2026.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

3.6 Answering RQ1

Having analyzed both theoretical research and practical implementations, we revisit our first research question: *What test strategies do existing cognitive architectures follow?* We predicted that different approaches to intelligence would lead to rather different test strategies across cognitive architectures. This was not entirely the case for the two architecture implementations inspected as part of our exploratory analysis. AERA and OpenNARS follow a similar high-level pattern using unit tests for specific functionalities, and task-environments for ability-oriented demonstrations and teaching purposes. However, the absence of a shared task theory means that while the high-level structure is similar, specific task-environments cannot be easily modified or compared across systems.

This highlights a fundamental tension in AI evaluation: while abstraction such as the ARC-AGI benchmark enables comparison, it risks reducing intelligence to a metric that can be optimized without genuine understanding. The task-environments developed in this work address this by taking a requirement-driven approach: deriving task-environments directly from a cognitive architecture's specification of intelligence, and documenting them in a way that serves as a lightweight stepping stone toward a more formal task theory.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 4

Methodology

With regard to our second research question in Section 1.2, we hypothesized that small task-environments can be derived from AERA's intelligence requirements and implemented as tests. In this chapter, we describe our three-fold approach to evaluating this hypothesis. First, we wrote preliminary tests based on Replicode's technical specification. Then, we mapped AERA's requirements for intelligence to their corresponding mechanisms in the form of a table. Finally, we implemented tests derived from this table while following predefined design principles. We conclude the chapter by relating our approach to established evaluation frameworks and discussing threats to its validity.

4.1 Preliminary testing

The Replicode technical report consists of a main part describing the higher-level constructivist programming paradigm as well as multiple annexes [20]. Writing preliminary tests against Annex 1 and Annex 2, the specifications of the Replicode language and its executive, proved especially insightful for three reasons: (1) it allowed us to explore different testing approaches for more sophisticated requirement-driven tests later on, (2) it allowed us to identify which parts of the syntax specification were still correct and where the report was outdated, and (3) it allowed us to better understand the workings and motivations behind existing task-environments created for test purposes.

We therefore systematically created tests for Annex 1 and Annex 2 of the Replicode technical report, with the main part and other annexes being tested sporadically. We highlighted and informally documented notable findings in the report PDF directly, with the following color system: Green means the specification was verified, yellow implies a minor change (e.g., parameter order was switched), and red signals that the specification was proven to be outdated or incorrect. The color blue was used for unverifiable statements such as descriptions. An example is shown in Figure 4.1, where the syntax specification of the built-in class `grp_view` is colored and annotated. The entire annotated PDF,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

as well as all preliminary test files, are included in the repository for this work. Test artifacts are documented in more detail in Chapter 5.

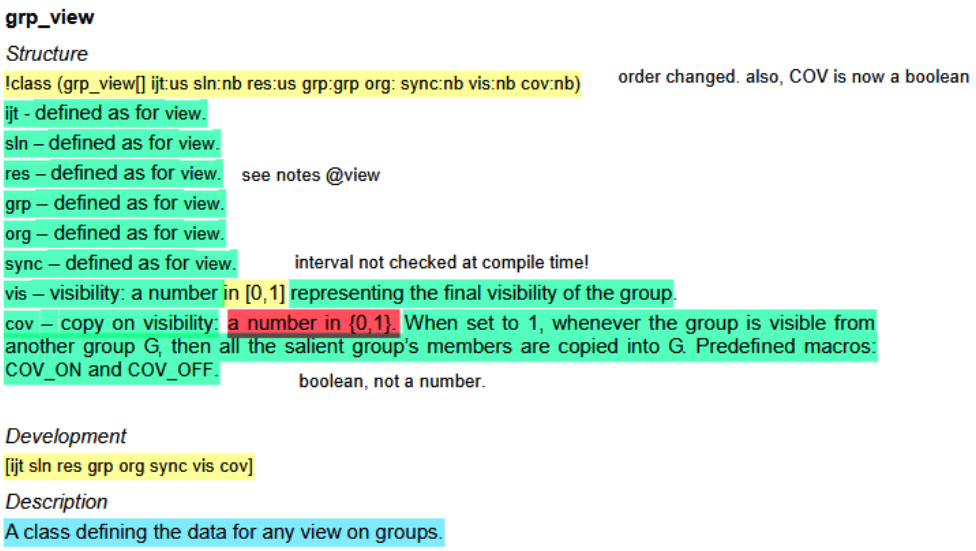

Figure 4.1: Annotations for the built-in class *grp_view*.

Each test created as part of preliminary testing consists of a Replicode/TXT file pair. The Replicode file represents the input, i.e., the seed with which AERA is run. The TXT file contains the expected results, which are matched against AERA's decompiled output of all created Replicode objects. Note that while this input-output structure resembles black-box testing, expected results are defined with knowledge of AERA's internal processes and object structure. We will discuss this structure more in Section 4.3, where we outline our test design principles for more complex requirement tests. But first, we describe how we collected these requirements in Section 4.2.

4.2 Intelligence requirement table

AERA's intelligence requirements were collected from multiple sources, including the methodology section on the OpenAERA web site [22], lecture notes from the course *Empirical Reasoning Systems* at Reykjavik University [27], the technical reports on AERA [21] and Replicode [20], and papers on specific requirements [28, 26, 25].

The resulting table was restructured several times. Requirements were merged where two names referred to the same property, and collapsed where one requirement was a side-effect of another. The table also documents for each requirement how it is achieved by AERA, in order to derive subsequent test cases. The full table consists of 9 major and 32 minor requirements, and would be too long to present in this section. It is documented in its entirety in Appendix A.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

4.3 Test design

With the intelligence requirement table fully specified, we now shift our focus towards **RQ2**, for which the goal is to examine which requirements can be tested in small task-environments. Our analysis of how AERA and Replicode achieve different requirements yielded specific mechanisms to test. Some of these mechanisms are modular enough to be covered to some extent by basic syntax tests, allowing us to reuse test cases created as part of Section 4.1. For more complex properties, three base tasks were created, by either reusing existing task-environments or creating novel ones. These base tasks are:

1. *bathtub*: The agent is tasked with filling a bathtub to a certain water level by controlling a valve.
2. *hand-grab-sphere*: The agent is tasked with moving a sphere to a certain position by controlling a hand.
3. *ball-external*: The agent is tasked with moving a ball to a certain position by controlling its speed.

These base tasks were then modified to test specific functionalities of AERA, while observing the following design principles:

- The Replicode/TXT input-output file pair structure established in Section 4.1 shall be followed. Expected outputs shall be derived with knowledge of AERA's internal processes and object structure. Therefore, although test cases retain a black-box input-output format, our approach is closer to gray-box testing.
- Each test case shall have a structured header in its Replicode input file describing the key properties of the created task-environment, including its base task and how it differs from it, which actions are available to the agent, and how AERA's parameters are changed from the defaults. This ensures that the intent of more complex tests can be understood at a glance.
- For each test, findings shall be reported in a test report document. This includes found bugs in AERA's implementation, errors in the documentation, or limitations or side notes of the test itself.
- With Visual Studio being AERA's main development environment, all tests shall be runnable using the integrated test explorer.
- While simply looking at the pass/fail status of tests in the test explorer is convenient, it is also less informative than inspecting the test files in more detail, as key information about the what, why and how of testing is lost. Therefore, both input and output files shall contain comments on what is happening in the seed code and why we look for certain expected results specifically.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

- Expected results shall be as robust as possible. As specific lines from the decompiled output are checked, these shall be abstracted to the point where the intention and validation of the test is still clear, but minor changes in the decompiled output format do not break all tests.

We illustrate these principles with two test snippets for Requirement 1.2 (acting upon the physical world). AERA achieves this requirement by calling commands on devices. There are two types of devices: internal and external.

- The internal device (i.e., the Replicode executive) accepts commands like modifying the confidence in a fact or injecting new knowledge. Internal commands are finite in number, and were already tested as part of preliminary testing in Section 4.1. We were therefore able to reuse certain tests for this requirement. These tests are basic syntax tests, meaning they do not follow a base task. Listing 4.1 shows the first lines of the test `mod`, which checks the internal command *modify*.
- External devices (e.g., a hand) can execute commands like moving, grabbing an object or turning a valve. To verify correct behavior of calling multiple commands on external devices, basic syntax tests do not suffice. Listing 4.2 shows the first lines of the test `bathtub`, which tests the use of external commands to achieve a simple goal.

Other requirements were tested analogously, by combining basic syntax tests with modified versions of one of the three base tasks.

Listing 4.1: First lines of `mod.replicode`

```
1  ; REQUIREMENT:   1.2 - Acting
2  ; PARAMETERS:    -
3  ; BASE TASK:     -
4  ; ACTIONS:       -
5
6  ; This is a basic syntax test for the icmd MODIFY.
7
8  ; M O D
9  ; !dfn (_mod : :nb)
10 ; !def (mod args) (icmd _mod args)
11
12 ; mod values
13 pgm0:(pgm |[] |[] |[] []
14     (mod [this.vw.act -0.1])
15     (mod [this.vw.sln 0.1])
16 1) |[]
17
18 ...
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Listing 4.2: First lines of `bathtub.replicode`

```
; REQUIREMENT:  1.2 - Acting
; PARAMETERS:   -
; BASE TASK:    bathtub
; ACTIONS:      open_valve / wait

; This is the base BATHTUB task.

; ============================
; CONSTANTS
; ============================
!def WL_EMPTY 0              ; empty bathtub
!def WL_FULL 100             ; full bathtub
!def WL_PERFECT 50           ; the water level we want to achieve
!def WL_DELTA 10             ; by how much the water level increases
   each sampling period

!def V_CLOSED 0              ; valve closed
!def V_OPEN 1                ; valve open

!def DRIVE_START 300ms       ; when to start injecting the drive
!def TIME_LIMIT 700ms        ; how long the agent has to achieve the
   goal. minimum time for task success is 600ms.

!def V_INIT V_CLOSED         ; initial value - valve
!def WL_INIT WL_EMPTY        ; initial value - water level

; ============================
; ENTITIES & ONTOLOGIES
; ============================
bathtub:(ent 1) [[SYNC_ONCE now 1 forever root nil]]
water_level:(ont 1) [[SYNC_ONCE now 0 forever root nil]];
valve_status:(ont 1) [[SYNC_ONCE now 0 forever root nil]];

; ============================
; HEARTBEAT
; ============================

; This is used below in pgm_inject_drive, to continuously inject our
   goal
b:(ent 1) [[SYNC_ONCE now 1 forever root nil]]
b_is_a_bathtub:(mk.val b essence bathtub 1) |[]
f_b_bathtub:(fact b_is_a_bathtub 0s:0ms:0us GIGASEC 1 1) [[SYNC_AXIOM
    now 1 forever stdin nil]]
...
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

4.4 Evaluation against related frameworks

To reflect on the intelligence-requirement-driven design described above, we analyze the resulting test suite using the two frameworks introduced in Sections 3.3 and 3.4: Hernández-Orallo's guidelines for ability-oriented testing and Thórisson et al.'s requirements for a task theory. This external standpoint lets us assess the extent to which our test suite follows these guidelines and requirements, in Sections 5.4 and 5.5. Our tests do not aim to fully satisfy either; rather, we use them as a lens to assess the completeness and principled design of the test suite.

4.5 Threats to validity

We conclude this chapter by discussing threats to the validity of our approach that are independent of any particular implementation or test suite.

Subjective judgment in deriving task-environments. While requirements for intelligence can be systematically collected and mapped to corresponding mechanisms, the specifics of the derived task-environments are ultimately left to the developer writing the tests. As requirements are formulated in natural language, the extent to which a test must satisfy a given requirement in order to validate it is inherently subjective.

Insufficient documentation. If the cognitive architecture under test lacks sufficient documentation of its requirements or its implementation, deriving meaningful tests is significantly hindered, if not impossible.

Maintainability difficulties. Our approach relies on inspecting the inner workings of a cognitive architecture. While this works well for stable versions of a system, systems under active development may require frequent updates to the test suite in order to keep expected test results consistent with the evolving implementation.

This concludes the methodology chapter. The next chapter documents the results of our test implementation, including artifacts, strengths, and limitations.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

CHAPTER 5

Results

This chapter presents the results of the testing process. It describes the artifacts in the accompanying GitHub repository, including the complete test report provided in Appendix B. It then examines the strengths and limitations of the concrete test implementation and reviews to what extent requirements for ability-oriented testing and a lightweight task theory were satisfied. Broader reflections on our testing approach are the topic of Chapter 6.

5.1 Test artifacts

The GitHub repository created as part of this work [23] was forked from the OpenAERA repository[1] on June 14, 2025. The Visual Studio solution with AERA's source code is unchanged besides the addition of the new project *test*, whose file structure is shown in Figure 5.1.

The *test* project contains the following files:

1. *tests/* is a sub-folder containing all created Replicode/TXT input-output pairs, including preliminary tests in *technical-report* and requirement-based tests in *intelligence-requirements*. The files *std.replicode* and *user.classes.replicode* serve as header files imported in each test.

2. *tests/settings_template.xml* contains the default parameters passed to AERA. Test cases specify if and how their parameters differ from these defaults.

3. *readme.md* contains instructions on how to set up, run, and explore test results inside Visual Studio.

[1] IIIM-IS/AERA on GitHub, accessed 12 August 2026.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

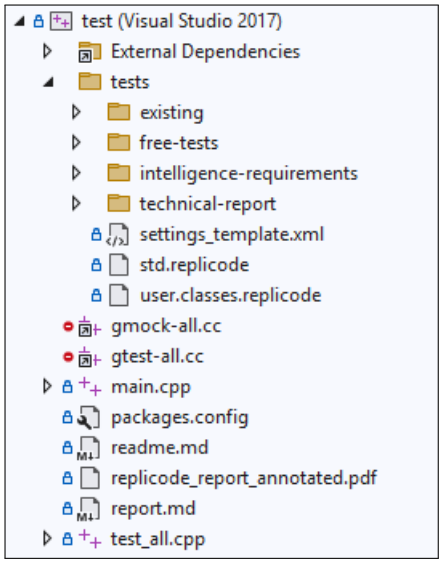

Figure 5.1: Folder structure of the *test* project.

4. *replicode_marked_annotated.pdf* contains the annotated Replicode technical report, as described in Section 4.1.

5. *report.md* serves as the main test report. It maps each intelligence requirement to specific tests and documents a short description and relevant findings for each test case. Appendix B consists of this report.

6. *test_all.cpp* is the C++ test script handling test execution.

Figure 5.2 shows the Visual Studio Test Explorer after successful execution of all 57 intelligence requirement tests. The explorer allows tests to be run individually, and displays error logs for failed tests. A full run of all tests takes approximately fifty seconds on an AMD Ryzen 7 3700X processor.

5.2 Strengths of the implementation

The Replicode/TXT input-output pair format was followed throughout all tests, leveraging the flexibility and reusability that specification-based testing provides [12]. This structure was already used in existing AERA task-environments, where output was inspected manually in either raw format or via the AERA Visualizer tool. Our test framework automates this evaluation.

Several aspects of our tests support future development and documentation of the AERA system. File headers and in-line comments improve readability and maintainability across the newly created task-environments. The dedicated *remarks* column in the test report highlights potential bugs, outdated documentation, and particularly interesting test cases.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

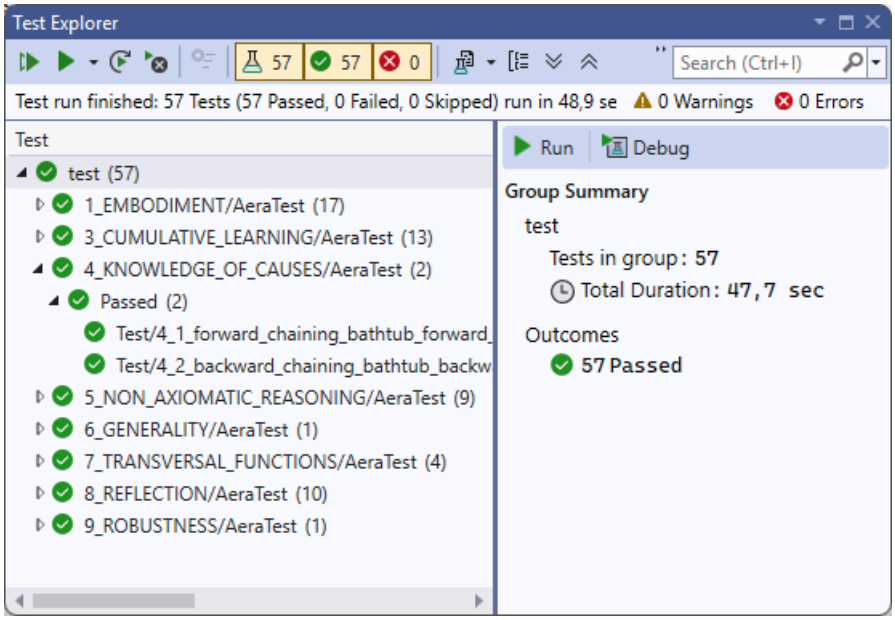

Figure 5.2: The Visual Studio Test Explorer displaying test results.

For example, the basic syntax tests for Requirement 1.1 document several modifications to the syntax of Replicode objects (*fact*, *goal*, *prediction*), and a potential bug was found during tests for Requirement 3.1 (*bathtub-hgs-parallel*).

The test framework is built entirely in Visual Studio, with no modifications to the original AERA code base and only one external dependency, the *gtest* framework, allowing for quick test execution and analysis using the built-in test explorer.

During preliminary testing, certain AERA features exhibited nondeterministic behavior. Affected parts were commented out in the seed and annotated with a remark to ensure consistent output (e.g., Requirement 8.1: *anything-as-input*). All requirement tests produced deterministic results across ten sequential runs.

A requirement-based test framework is only as good as its requirements. In our case, the intelligence requirement table partially covered all main sections of the Replicode technical report [20], with special focus on Section 6 – *High-Level Reasoning*, Annex 1 – *Replicode Specification*, and Annex 2 – *Specification of the Executive*. We further partially verified expected behavior of the AERA system as specified in other research papers [21, 26, 28]. We do not claim full coverage of AERA's functionality, nor that our tests fully cover our own specification. Therefore, both the intelligence requirement table and the test framework are designed to support future additions and modifications.

The requirement-driven approach allowed us to isolate specific properties of AERA. While tasks like *bathtub* exercise multiple cognitive abilities simultaneously, targeted lines in the expected output file allow us to filter the full decompiled output for relevant objects. As

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

a result, even when AERA achieves a task goal, a test may fail for requirement-specific reasons, for example, if control values are not updated as expected.

5.3 Limitations of the implementation

Although we derived 57 distinct tests from the intelligence requirement table, some requirements have no corresponding tests. This is the case for one of four reasons:

1. The requirement is not covered by the current OpenAERA implementation, either because the corresponding mechanism was only described in theory, or because it was implemented but is not stable enough for the main branch. This applies to Requirements 9.2 – *Scalability* and 5.6 – *Analogy*, respectively.

2. The requirement is so fundamental to AERA that it is satisfied by design. While all other tests capture this property to some extent, testing for it specifically is not meaningful, as it cannot fail unless AERA's design is fundamentally changed. This applies to Requirements 2 – *Constructivism*, 6.1 – *Flexibility*, 6.3 – *Constraint diversity*, and 6.4 – *Goal diversity*.

3. The requirement is too broad for specific tests. This applies to Requirements 9.3 – *Predictability*, 9.4 – *Reliability*, and 9.5 – *Trustworthiness*. Note, however, that ensuring AERA behaves in a predictable, reliable, and trustworthy manner is precisely the purpose of all other tests.

4. The requirement is achieved through the same mechanisms as another requirement. This applies to Requirement 8.2 – *Self-programming*, for which we refer to tests for Requirements 5.5 – *Induction* and 3.1 – *Multitask learning*.

One of our test design principles was to make expected output files as robust as possible against changes in AERA's decompiled output format. For example, we removed the detailed object identifiers, which are an incrementing counter of every object in the current AERA instance. Abstracting the output too aggressively would reduce expressivity. This conflict is illustrated in Listing 5.1, where the creation of a *success* object as evidence of goal achievement is checked. Identifiers such as `mk_val74` are brittle, as they are created at runtime and small changes to object creation in AERA would change the identifier to a different character string. Without the identifiers, however, expressing relations between objects in the expected output file would be hindered.

Listing 5.1: Expected results of the *bathtub* task.

```
1 (...)
2 ; then, we verify that the goal state was achieved
3 mk_val74:(mk.val b water_level 50 1) |[]
4 fact_516:(fact mk_val74 0s:900ms:0us 1s:0ms:0us 1 1) []
5 success17:(success fact_508 fact_516 mk_rdx_510 1) |[]
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

While our framework supports automatic test execution, meaningful examination of tests or debugging failures in particular still require familiarity with the system's internals. This is partly a consequence of AERA's error logs being limited and occasionally obscure, and partly inherent to the complexity of writing Replicode seed files manually. Yet it is also consistent with our stated goal of supporting both convenient pass/fail status checks and deeper manual exploration.

The current implementation does not support parallel test execution. This was not necessary for our work as a full test run takes only fifty seconds. However, if more resource-intensive tests were added, test parallelization could be achieved by refactoring the way the parameter file is handled and starting multiple AERA instances at once.

5.4 Reviewing requirements for ability-oriented testing

In Section 3.3, we presented Hernández-Orallo's guidelines for future AI benchmark competitions in the context of ability-oriented testing [10]. In the following, we assess the extent to which these guidelines are reflected in our tests.

Guidelines 1, 5, and 6 are not applicable, as our work does not define a competition but a test library for a specific version of a cognitive architecture.

Guideline 2 recommends defining the set of possible tasks and their distribution. In our case, tasks are specified informally in the test report and in comments within the test input files.

Guideline 3 requires a scoring function that produces comparable, bounded values across tasks. Our framework relies on pass/fail evaluation, with more detailed analysis left to manual inspection.

Guideline 4 suggests identifying similarities between tasks. We address this by defining three base tasks and informally describing how derived tests relate to them.

Guideline 7 calls for analysis beyond aggregated scores. We support this through detailed task documentation, enabling evaluators to move beyond pass/fail results and examine which internal mechanisms are required and successfully realized for different tasks.

5.5 Reviewing requirements for a task theory

One of our goals was to document tests in a way that serves as a lightweight stepping stone towards a more formal task theory. In the following, we discuss the extent to which the documentation of our testing framework and created task-environments satisfy the requirements for a task theory introduced in Section 3.4 [30, 29]:

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

1. **Task comparison.** We support informal comparison of tasks to their base task through text-based descriptions in the input files.
2. **Task abstraction and concretization.** This is not supported in the current implementation and would instead require the creation of new tasks. In principle, parameterization of tests could enable different levels of abstraction.
3. **Estimation of time and resource requirements for task completion.** This is not supported, as our tests are simple enough to result in approximately constant resource usage. In principle, this would be possible and sensible, since most tests run in diagnostic mode, which allows AERA to explore all available causal paths before committing to an action.
4. **Characterization of task complexity.** Not supported in the current implementation.
5. **Easy task construction through composition, decomposition, and scaling.** This is partially supported in that some tasks are the results of composition and scaling of base tasks.
6. **Procedural generation.** This was explored experimentally but not adopted due to the difficulty of procedurally generating meaningful and syntactically correct Replicode objects.
7. **Transparency to analysis.** All task-environments are available in the GitHub repository [23], and the test report is documented in Appendix B.

Task-environments created as part of such a task theory should support the following properties [29]:

1. **Determinism and partial stochasticity.** Not satisfied, as all task-environments we created are fully deterministic.
2. **Ergodicity.** Not satisfied, as agents are not provided second chances in any of our tests unless the point of the test is precisely that.
3. **Controllable continuity.** Not satisfied, as all variables in our task-environments are discrete.
4. **Asynchronicity.** Satisfied, as commands in AERA are asynchronous by design, and so are our mocked environments.
5. **Dynamic or static state handling.** Partially satisfied; most tests are static, while some emulate dynamic environments (e.g., *ball-external*).
6. **Varying degrees of partial observability.** Partially satisfied; the amount of observed information is determined in the seed.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

7. **Varying degrees of controllability.** Partially satisfied; some tests emulate commands called by the agent to fail.
8. **Multiple parallel causal chains.** Satisfied; and there is a test for this specific property, see *bathtub-hgs-parallel*.
9. **Varying number of agents.** Not satisfied, as all our tests involve a single agent.
10. **Periodicity.** Not satisfied, as tasks are too limited in scope to require learning of temporal or structural patterns.
11. **Repeatability.** Satisfied, as all tests are fully repeatable.

5.6 Answering RQ2

Having implemented and analyzed a requirement-based test framework, we revisit our second research question: *Which of AERA's special properties can be tested in small, requirements-driven task-environments?* We hypothesized that treating AERA's intelligence requirements like a software specification would allow specific cognitive processes to be evaluated in small task-environments.

We collected 9 major and 32 minor intelligence requirements for AERA, largely covering the Replicode technical report as well as other research papers. Of these 32 minor requirements, 20 were tested in 57 small task-environments or basic syntax tests. The remaining 12 are either indirectly addressed through other tests, satisfied by AERA's design, or not yet testable due to incomplete implementation. The main advantage of small requirements-driven task-environments is the ability to isolate specific cognitive properties via targeted lines in the expected output. The main constraint is robustness: runtime-generated identifiers in the expected output files are brittle and may cause tests to break if AERA's implementation changes.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 6

Discussion

Building on the test framework developed for our AERA case study, this chapter discusses the broader implications of the requirements-driven testing approach and its extensibility to other cognitive architectures.

6.1 Requirement-driven tests for cognitive architectures

Our requirements-driven testing approach offers three central benefits over existing methods:

Evaluation. Test results provide both a concise overview and the ability for detailed inspection of the system's current performance across various mechanisms. There is a clear mapping from intelligence requirement to cognitive mechanism to corresponding tests, making it possible to identify not only whether a system passes or fails at a task, but which specific mechanisms are responsible. The remarks in the test report flag outdated or incorrect documentation, contributing beyond pass/fail evaluation. This level of traceability is not offered by demos or general-purpose benchmark tests.

Development. Future development is facilitated by an extensible test library that provides regression coverage. When existing features are refactored or new ones introduced, the suite can be rerun to confirm that previously valid behavior is preserved.

Education. Understanding the purpose of existing test tasks typically requires familiarity with the system's internals. Annotated test input files complement the system documentation by making the targeted requirement, base task, and expected behavior explicit at the top of every file, lowering the barrier for developers new to the system.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

While our approach uses task-environments to test features of intelligence, it follows an ability-oriented design. By checking for specific internal processes of the system under test as part of the expected output, we evaluate not only *whether* the system achieves a goal, but *how* it achieves it and what is learned in the process. Accordingly, the focus is not on the system's actions, but on the internal mechanisms that lead to them. This aligns with Hernández-Orallo's motivation for ability-oriented testing, as discussed in Section 3.3.

A central aspect of evaluation is comparison across systems. The current approach does not directly support this. Comparing two cognitive architectures would require them to share the same requirements, which is unlikely given that architectures are grounded in differing theories of intelligence. What can be compared, however, are overlapping requirements, even if they are realized through different mechanisms. A principled task theory, as described in Section 3.4, would further assist this, enabling different cognitive mechanisms for the same requirement to be evaluated in distinct but comparable task-environments. Achieving this in practice remains an open problem for the field.

There is an apparent tension between using specification-based testing (i.e., black-box testing) and evaluating the internal mechanisms of the system under test. While our approach is specification-based, expected outputs are defined with knowledge of the system's internal processes and object structure. Consequently, test cases follow a black-box input-output format, but the reliance on internal system knowledge places the approach closer to gray-box testing. Therein lies both the greatest strength and limitation of our approach: Using inner workings as output allows us to inspect specific inner mechanisms, but risks tests to break when the internal representation of the system changes.

6.2 Generalizability to other cognitive architectures

The requirements-driven testing approach developed for AERA is not specific to that system. Any cognitive architecture satisfying the following three conditions can be tested using the same method:

Explicit design methodology. Intelligence requirements can only be collected if the architecture's design principles are explicitly stated. For AERA, these were documented across several publications, the architecture's website, and teaching material.

Software implementation. Tests must be written against a concrete implementation. An architecture may have multiple implementations, but as long as intelligence requirements can be mapped to specific mechanisms, the approach applies. Sufficient documentation of the implementation is also important. For AERA, the Replicode technical report was essential for identifying which mechanisms correspond to which requirement.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

White-box design. The approach evaluates not just whether a system achieves a goal, but how it does so. This requires the ability to inspect internal processes, for example through log traces or decompiled output. Black-box systems do not expose the internal processes necessary for requirement-driven evaluation.

6.3 Answering RQ3

Having analyzed the generalizability of our approach to other cognitive architectures, we return to our third research question: *To what extent can the testing approach developed for AERA in RQ2 be generalized to other AGI-aspiring systems?* We hypothesized that the approach is generalizable by substituting AERA's requirement list with that of other cognitive architectures. This is largely correct, but incomplete. As discussed in Section 6.2, both an explicitly documented design methodology and a white-box software implementation are required.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 7

Conclusion

In this thesis we investigated the feasibility of treating a cognitive architecture's intelligence requirements as a software specification, and deriving concrete test cases from it systematically. The motivation was a gap in existing evaluation methods: cognitive architectures define explicit intelligence requirements, yet these are rarely reflected in how their implementations are tested. We chose AERA for a case study, collected its intelligence requirements, and derived a test suite for the system.

7.1 Key contributions

We gave an overview of existing evaluation methods for cognitive architectures. It showed that different systems follow a similar high-level structure, with unit tests for specific functionalities and task-environments for broader intelligence demonstrations. However, tests are usually constructed ad hoc and lack a direct connection to the system's intelligence requirements.

Building on this, we developed a requirements-driven testing approach, and implemented and analyzed this approach for AERA. The resulting test suite provides a direct mapping from intelligence requirement to tests, and supports the AERA project in three areas: it enables traceable evaluation of specific cognitive properties, provides regression coverage for future development, and lowers the barrier for developers new to the system through annotated task-environment Replicode files.

Regarding generalizability, we determined that the approach extends to any cognitive architecture satisfying three conditions: an explicitly documented design methodology, a concrete software implementation, and a white-box design that exposes internal processes for inspection.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

7.2 Future work

There are immediate ways to further explore our approach, and more complex issues that might require long-term investigation. For the final section of this thesis, we list some options in order of complexity, starting with the most straightforward.

Testing is never fully complete. Since our test framework for AERA is easily extensible, the test suite could be expanded to cover more areas of AERA's functionality. While it is arguably impossible to fully verify certain intelligence requirements, each additional test strengthens confidence in the system's correctness.

Our approach could be applied to another cognitive architecture that fulfills the aforementioned generalization criteria. Soar, NARS, and LIDA should all be possible candidates, each posing different challenges. For example, the federated development of LIDA could make the mapping from intelligence requirement to specific mechanisms more difficult than it was for AERA.

We identified the brittle identifier problem as a negative side effect of inspecting internal processes. For AERA, one direction would be to transform the decompiled output into a more maintainable format, removing the reliance on runtime-generated object identifiers.

The overarching issue is how to make explainable AI systems easier to test in general. A principled task theory would provide significant benefits in this regard, as discussed throughout this thesis. Equally important is maintaining comprehensive and up-to-date documentation of cognitive architectures, without which requirements-driven testing cannot be applied effectively.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

APPENDIX A

Intelligence Requirement Table

Requirement	AERA Mechanism	Sources
1 Embodiment: Required to find out what the current status of the world is by generating models of it, with respect to one's goals. Requirement for 4 - Knowledge of causes.		[22]
1.1 Sensing: Perceiving and making measurements of the physical world through sensors.	Inputs coming from the environment (technically, data produced by the I/O devices) come in the form of facts for which the time interval is reduced to a single point in time (the time the input was produced) and for which the confidence value is one (since the fact represents an observation in the environment).	[27] [20] - 6.1
1.2 Acting: Acting upon the physical world to achieve goals.	AERA acts upon the physical world by calling commands on devices. There are two types of devices: internal (the executive, e.g.: inj, eje, mod, set, ...) and external (using custom commands).	[27] [20] - 2
2 Constructivism: Required so that a system can autonomously amass the vast majority of its knowledge (except for a small seed), in a way that infuses meaning into information structures. "Autonomous" means that this can be done without interference / help from others / the need to "call home".		[22, 27]
2.1 Generate knowledge autonomously: The system acquires the vast majority of its knowledge on its own (except for a small seed).	By design. This is so central to AERA that all tests can be thought of as capturing this to some extent.	[27, 26]
2.2 Manage knowledge autonomously: The system manages its own growth on its own.	See 2.1.	[27, 26]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Requirement	AERA Mechanism	Sources
2.3 Use knowledge autonomously: The system may also change its own architecture over time, due to experience and learning.	See 2.1.	[27, 26]
3. Cumulative Learning: Required to unify new information and knowledge that is already acquired, in a coherent, efficient and effective manner. Unifies several separate research tracks.		[22]
3.1 Multitask Learning: The ability to learn more than one task, either at once or in sequence.	Programs and models are reactive and triggered whenever there is a left-side match AND related saliency/activation values are high enough.	[27] [21] - 6.2 [20] - 2
3.2 Online Learning: The ability to learn continuously, uninterrupted, and in real-time from experience as it comes, and without specifically iterating over it many times.	AERA can use new knowledge whenever it comes along.	[27] [20] - 2
3.3 Lifelong Learning: The ability of an AI system to keep learning and integrating knowledge throughout its operational lifetime: learning is "always on".	AERA can use new knowledge (or react to changes in saliency/activation) even while/after pursuing a more complex task.	[27] [20] - 2
3.4 Robust Knowledge Acquisition: The antithesis of which is brittle learning, where new knowledge results in catastrophic perturbations of prior knowledge (and behavior).	Through control values (saliency, activation, resilience) that control which models are more reliable than others.	[27] [21] - 6.2
3.5 Transfer Learning: The ability to build new knowledge on top of old in a way that the old knowledge facilitates learning the new. While interference/forgetting should not occur, knowledge should still be defeasible.	In AERA, models exist separately from tasks, so any learned model can be used for any task.	[27, 28]
4. Knowledge of Causes: Required to guide action. Without knowledge about causal relations it is impossible to get anything systematically done. The only known method for creating causal models is combined forward-backward chaining.		[27]
4.1 Forward Chaining: Required to produce predictions. Uses matching to produce what might happen next, after a particular state is reached, starting with existing data, until an endpoint (typically a goal) is reached.	Concatenated deduction (5.3).	[27] [20] - 6.3 [21] - 7.3.2
4.2 Backward Chaining: Required to produce plans. Starts with a given goal or state to be achieved, and proceeds through matching to produce what could possibly have been the state just prior to that particular state.	Concatenated abduction (5.4).	[27] [20] - 6.3 [21] - 7.3.2
5. Non-axiomatic Reasoning: Required to operate on knowledge in worlds where the axioms are unknown, not guaranteed, and/or fallible.		[22]
5.1 Defeasible Reasoning: Reasoning where the system-produced output may be found to be incorrect, at any time, given additional data, reconsideration of background assumptions or discovery of logic errors.	Models that have repeatedly made poor predictions are removed from AERA. Activation values are used to monitor the system, to first deactivate and then entirely remove models/programs.	[27] [20] - 6.3

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Requirement	AERA Mechanism	Sources
5.2 Pattern Matching: Pattern matching is the main method that reasoning systems use in their operation.	Pattern matching is central in AERA, where all existing object classes can be matched.	[27] [20] - 3
5.3 Deduction: Figuring out the implication of facts (or predicting what may come). Producing implications from premises.	In Replicode, deduction is the production of predictions given an input fact (actual or predicted).	[27] [20] - 6.3
5.4 Abduction: Figuring out how things came to be the way they are (or how particular outcomes could be made to come about, or prevented).	In Replicode, abduction refers to the production of sub-goals, given an input goal.	[27] [20] - 6.3
5.5 Induction: Figuring out the general case. Making general rules from a (small) set of examples.	Replicode offers three ways of acquiring models/states via three Targeted Pattern Extractors (TPX): Goal TPX, Prediction TPX, Change TPX.	[27] [20] - 6.4
5.6 Analogy: Figuring out how things are similar or different. Making inferences about how something X may be through a comparison to something else Y, where X and Y share some observed properties.	The analogy process happens by comparing situation-goal pairs with the instantiated LHS and RHS of Mreq during backward chaining, identifying matching and non-matching patterns.	[27] [25] - 6.4.2
6. Generality (Domain Independence): Required as all requirements in this list must be independent of the subject of learning. Any system X that exceeds system Y on one or more of the points below is considered more general than system Y.		[22]
6.1 Flexibility: If a system X can operate in more diverse task-environments than system Y, system X is more flexible.	By design: "The architectural principles do not make any assumption on the target domain."	[27] [21] - 6.2
6.2 Solution Diversity: If a system X can reliably generate a larger variation of acceptable solutions to problems than system Y, system X is more powerful.	See 6.1.	[27] [20] - 6.5
6.3 Constraint Diversity: If a system X can reliably produce acceptable solutions under a higher number of solution constraints than system Y, system X is more powerful.	See 6.1.	[27] [20] - 6.5
6.4 Goal Diversity: If a system X can meet a wider range of goals than system Y, system X is more powerful. Multiple goals should also be achieved concurrently.	The main contributor is the Reaction activity. The parallel data-driven execution model allows the pursuit of multiple goals concurrently.	[27] [21] - 6.2
7. Transversal Functions: The system must have pan-architectural characteristics that enable it to operate consistently as a whole, to be highly adaptive yet robust in its own operation across the board, including metacognitive abilities.		[22]
7.1 Transversal Handling of Time: Time is a semantic property, and the system must be able to understand, and learn to understand, time as a real-world phenomenon in relation to its own skills and architectural operation.	Time is relevant to every object; completely ingrained in the system. It was changed from exact moments to a range.	[27]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Requirement	AERA Mechanism	Sources
7.2 Transversal Resource Management: Resource management involves management of knowledge, time, energy, and computing power.	Resources are self-monitored. If AERA starts taking too long to process reductions, three measures can be taken (e.g. raising group activation/saliency thresholds).	[27] [20] - 7
7.3 Preemptibility: The system has to be able to suspend its current operation in favor of more urgent tasks, and resume the former when possible, if it is still desired.	Attention control in AERA is implemented via the "auto-focus", which is recomputed every cognitive cycle, directing the system's attention with regards to its internal input.	[21] - 8.2.2 [20] - 6.5
8. Reflection: Required for a system to learn, over time, about its own fallacies, shortcomings, and lack of knowledge.		[22]
8.1 Meta-Cognition: The ability of a system to evaluate itself and reason about itself.	This is central to AERA by design. Notably, Notifications allow AERA to evaluate itself, and any reductions and knowledge about knowledge serve as input to reason about itself.	[27] [20] - 5
8.2 Self-Programming: The ability of a system to generate its own programs using fixed design-time principles while deciding the specific details at runtime based on its experience.	Achieved per design.	[27] [20] - 6
9. Robustness: When the system achieves consistently a set of goals within some constraints, it shall continue to do so despite novelty occurring in the environment, provided this novelty changes the initial goals and constraints only marginally.		[21]
9.1 Graceful Degradation: No general autonomous system operating in the physical world can perform flawlessly throughout its lifetime. The system must be constructed to minimize potential for catastrophic failure.	AERA's knowledge processing is organized around goals. With increased knowledge AERA gets closer to meeting its top-level drives. Furthermore, AERA can do reflection, so it gets better at evaluating its own performance over time.	[27]
9.2 Scalability: The system shall maintain a constant level of performance independently from the complexity of the environment and of the knowledge it has accumulated.	AERA identifies stable paths of subgoals/predictions, which are compacted for more efficiency. The original models are still saved in case the compact structure proves unstable.	[21]
9.3 Predictability: The ability of an outsider to predict the behavior of a controller based on some information.	By design. The purpose of testing is precisely to verify the behavior of AERA based on some information.	[27]
9.4 Reliability: The ability of a machine to always return the same — or similar — answer to the same input. Requires predictability.	By design. The purpose of testing is to verify that AERA behaves deterministically in test cases.	[27]
9.5 Trustworthiness: The ability of a machine's owner to trust that the machine will do what it is supposed to do. Requires predictability and reliability at multiple levels of operation.	By design. See 9.3 and 9.4.	[27]

Table A.1: Intelligence requirements and corresponding AERA mechanism.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

APPENDIX B

Test Documentation

Test Name	Description	Remarks
1.1 Sensing		
fact	Tests basic fact syntax.	(R1) The parameters after and *before* use timestamps, not microseconds.
anti-fact	Tests basic anti-fact syntax.	(R1) The parameters *after* and *before* use timestamps, not microseconds.
goal	Tests basic goal syntax.	(R1) The structure of the goal class has changed.
prediction	Tests basic prediction syntax.	(R1) The structure of the prediction class has changed.
ball-external	Tests if inputs from the (mocked) environment are handled correctly.	(R1) Since we are in a unit test environment, we use the existing mocked I/O device "ball".
1.2 Acting		
inj	Tests the icmd `_inj`.	(R1) In decompiled_objects, the name of the injected object does not match what is provided. (R2) If the object is injected to group "stdin", the object has no view in the decompilation file.
eje	Tests the icmd `_eje`.	(R1) This has never been used in existing Replicode files, and the test only partially covers what the icmd should be able to do.
mod	Tests the icmd `_mod`.	—
set	Tests the icmd `_set`.	(R1) The documentation states that -1 represents an infinite resilience, but this was changed to the keyword "forever".
new-class	Tests the icmd `_new_class`.	(R1) This has never been used in existing Replicode files, and the test only partially covers what the icmd should be able to do.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Test Name	Description	Remarks
del-class	Tests the icmd `_del_class`.	(R1) This has never been used in existing Replicode files, and the test only partially covers what the icmd should be able to do.
ldc	Tests the icmd `_ldc`.	(R1) This has never been used in existing Replicode files, and the test only partially covers what the icmd should be able to do.
swp	Tests the icmd `_swp`.	(R1) This has never been used in existing Replicode files, and the test only partially covers what the icmd should be able to do. (R2) Documentation states that only values in {0, 1} should be allowed as a parameter, but this is not the case.
prb	Tests the icmd `_prb`.	(R1) This has only ever been used in combination with "print", which works as expected.
stop	Tests the icmd `_stop`.	—
cmd	Tests basic command syntax.	(R1) The parameter `device:did` (as specified in the documentation) does not exist anymore.
bathtub	Tests the use of external commands to achieve a simple goal.	—
2.1 Generate knowledge autonomously		
No tests. Achieved per design; all other tests capture this to some extent.		
2.2 Manage knowledge autonomously		
No tests. Achieved per design; all other tests capture this to some extent.		
2.3 Use knowledge autonomously		
No tests. Achieved per design; all other tests capture this to some extent.		
3.1 Multitask Learning		
programs	Tests basic program syntax.	—
models	Tests basic model syntax.	—
mediation-mod	Tests the mediation of `mod` commands.	—
mediation-set	Tests the mediation of `set` commands.	(R1) Setting and averaging works, but illegal values are rounded to 0 or 1.
bathtub-hgs-sequence	Runs the bathtub and hand-grab-sphere tasks in sequence.	—
bathtub-hgs-parallel	Runs the bathtub and hand-grab-sphere tasks in parallel.	(R1) Goal of test was not achieved; enabling model M7 causes AERA to run indefinitely.
3.2 Online Learning		
bathtub-new-knowledge	Tests if the system can use new knowledge whenever it comes along.	—
3.3 Lifelong Learning		
bathtub-act-change	Tests the system's reaction to a control value change.	—
ball-external-new-models	Tests model creation from facts observed by an external device during a task.	—

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Test Name	Description	Remarks
3.4 Robust Knowledge Acquisition		
bathtub-success-rate-increase	Tests model success rate update after correct predictions.	—
bathtub-success-rate-decrease	Tests model success rate update after failed predictions.	—
bathtub-success-rate-reliable	Tests model success rate update on reliable models.	—
3.5 Transfer Learning		
sink	Tests the models of the bathtub task on a different but similar task.	—
4.1 Forward Chaining		
bathtub-forward-chaining	Tests the first part of planning: predictions.	—
4.2 Backward Chaining		
bathtub-backward-chaining	Tests the second part of planning: sub-goals.	—
5.1 Defeasible Reasoning		
bathtub-model-defeated	Tests the system's behavior after a model's prediction is defeated.	—
bathtub-sim-defeated	Checks output after a promoted simulation prediction is defeated.	(R1) This task-environment is incomplete, but it suffices for the purpose of this test.
5.2 Pattern Matching		
ptn	Tests basic pattern syntax.	(R1) Patterns cannot exist on their own and are therefore tested as part of other objects.
bathtub-pattern-matching	Tests various instances of pattern matching.	—
5.3 Deduction		
bathtub-deduction	Tests created predictions and their confidence values.	—
5.4 Abduction		
bathtub-abduction	Tests created sub-goals and their confidence values.	—
5.5 Induction		
hgs-GTPX	Tests the GTPX mechanism.	—
ball-PTPX	Tests the PTPX mechanism.	(R1) The PTPX creates several duplicate composite states and models.
ball-CTPX	Tests the CTPX mechanism.	(R1) The induced model should eventually get added to the blacklist, but this behavior could not be verified. (R2) `v7*100ms` seems to perform `(* ts nb)`, which according to the list of Gotchas in the OpenAERA readme[1] should not be allowed.
5.6 Analogy		
No tests. The analogy feature is not merged to the main branch.		
6.1 Flexibility		
No tests. Achieved per design; architectural principles make no assumption on the target domain.		

[1] IIIM-IS/AERA Gotchas on GitHub, accessed 12 August 2026.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Test Name	Description	Remarks
6.2 Solution Diversity		
solution-diversity	Tests if AERA can find multiple available plans to the goal state.	(R1) This task leads to a "prediction explosion" at 0s:500ms:0us, which can be inspected in the Visualizer after loading for a while.
6.3 Constraint Diversity		
No tests. Achieved per design; any Replicode object can be a solution constraint.		
6.4 Goal Diversity		
No tests. Achieved per design. See 3.1 for parallel and sequential goal achievement.		
7.1 Transversal Handling of Time		
bathtub-timestamps	Checks that the bathtub task also succeeds with different timestamps.	—
7.2 Transversal Resource Management		
bathtub-real-time	Checks the bathtub task in real-time mode.	(R1) If AERA does not complete planning in time, it does not commit to the deduction path it partially explored.
bathtub-perf	Checks the `perf` object created in real-time mode.	—
7.3 Preemptibility		
bathtub-hgs-preemptibility	Checks that when a more important goal is injected while another is being pursued, the former is pursued.	(R1) AERA is unable to complete this task, as it cannot find a plan in real time.
8.1 Meta-Cognition		
mk-rdx	Checks the notification `mk.rdx`.	—
mk-new	Checks the notification `mk.new`.	—
mk-low-sln	Checks the notification `mk.low_sln`.	—
mk-high-sln	Checks the notification `mk.high_sln`.	—
mk-low-act	Checks the notification `mk.low_act`.	—
mk-high-act	Checks the notification `mk.high_act`.	—
mk-low-res	Checks the notification `mk.low_res`.	—
mk-sln-chg	Checks the notification `mk.sln_chg`.	(R1) Expected results of this test are commented out due to inconsistent creation of mk.sln_chg objects
mk-act-chg	Checks the notification `mk.act_chg`.	(R1) Expected results of this test are commented out due to inconsistent creation of mk.act_chg objects
anything-as-input	Checks that different objects (e.g., models, programs, act/sln/res notifications) can be used as input for models.	(R1) CSTs and ICSTs occasionally cause read-access violations. (R2) Visualizer fails to parse `icmd` in models, even though this seems to work fine in the decompiled output.
8.2 Self-Programming		
No tests. See 5.5 for induction tests and 3.1 for basic model and program syntax tests.		

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Test Name	Description	Remarks
9.1 Graceful Degradation		
bathtub-hgs-ball-failure	Checks the system's output after failing to achieve several goals.	—
9.2 Scalability		
No tests. The compaction mechanism has not been implemented.		
9.3 Predictability		
No tests. The purpose of all other tests is precisely this.		
9.4 Reliability		
No tests. The purpose of all other tests is precisely this.		
9.5 Trustworthiness		
No tests. See 9.3 and 9.4.		

Table B.1: Intelligence requirement test report.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Overview of Generative AI Tools Used

The free version of Anthropic Claude (Sonnet 4.6) was used to assist with grammar checking, LaTeX syntax (especially for the tables in Appendix A and B), and refining the wording of specific sentences. No text passages were fully created by AI tools.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Figures

4.1 Annotations for the built-in class *grp_view*. 20

5.1 Folder structure of the *test* project. 26
5.2 The Visual Studio Test Explorer displaying test results. 27

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Tables

3.1 Four categories of artificial intelligence tests with examples. 12

A.1 Intelligence requirements and corresponding AERA mechanism. 42

B.1 Intelligence requirement test report. 47

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Bibliography

[1] Bernard J. Baars. Global workspace theory of consciousness: toward a cognitive neuroscience of human experience. pages 45–53. 2005.

[2] Jordi Bieger, Kristinn R Thórisson, Bas R Steunebrink, Thröstur Thorarensen, and Jona S Sigurdardottir. Evaluation of general-purpose artificial intelligence: why, what & how. *Evaluating General-Purpose AI*, 9, 2016.

[3] François Chollet. On the Measure of Intelligence. 11 2019.

[4] François Chollet. ARC-AGI-2 Overview. YouTube, 2025.

[5] François Chollet, Mike Knoop, Gregory Kamradt, and Bryan Landers. ARC Prize 2025: Technical Report. 1 2026.

[6] Stan Franklin, Tamas Madl, Sidney D'Mello, and Javier Snaider. LIDA: A Systems-level Architecture for Cognition, Emotion, and Learning. *IEEE Transactions on Autonomous Mental Development*, 6(1):19–41, 3 2014.

[7] Olivier Georgeon. Interview by Maximilian Pfeil. Personal interview, 2025.

[8] Olivier L. Georgeon, James B. Marshall, and Riccardo Manzotti. ECA: An enactivist cognitive architecture based on sensorimotor modeling. *Biologically Inspired Cognitive Architectures*, 6:46–57, 10 2013.

[9] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring Massive Multitask Language Understanding. 1 2021.

[10] José Hernández-Orallo. Evaluation in artificial intelligence: from task-oriented to ability-oriented measurement. *Artificial Intelligence Review*, 48(3):397–447, 10 2017.

[11] José Hernández-Orallo and David L. Dowe. Measuring universal intelligence: Towards an anytime intelligence test. *Artificial Intelligence*, 174(18):1508–1539, 12 2010.

[12] ISO, IEC, and IEEE. *Software and systems engineering: software testing. Part 1, Concepts and definitions*. 2013.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[13] Iuliia Kotseruba and John K. Tsotsos. 40 years of cognitive architectures: core cognitive abilities and practical applications. *Artificial Intelligence Review*, 53(1):17–94, 1 2020.

[14] Sean Kugele. Interview by Maximilian Pfeil. Personal interview, 2025.

[15] John E. Laird. Introduction to Soar. 5 2022.

[16] John E. Laird, Allen Newell, and Paul S. Rosenbloom. SOAR: An architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64, 9 1987.

[17] Shane Legg and Marcus Hutter. A Collection of Definitions of Intelligence. 6 2007.

[18] Shane Legg and Marcus Hutter. Tests of Machine Intelligence. 12 2007.

[19] Shane Legg and Joel Veness. An Approximation of the Universal Intelligence Measure. 9 2011.

[20] Eric Nivel and Kristinn R Thórisson. Replicode: A Constructivist Programming Paradigm and Language. *Technical RUTR-SCS13001, Reykjavik University School of Computer Science*, 2013.

[21] Eric Nivel, Kristinn R Thórisson, Harris Dindo, Giovanni Pezzulo, Manuel Rodriguez, C Corbato, B Steunebrink, D Ognibene, A Chella, and others. Autocatalytic endogenous reflective architecture. *Technical RUTR-SCS13002, Reykjavik University School of Computer Science*, 2013.

[22] OpenAERA. Methodology. https://openaera.org/methodology, 2021.

[23] Maximilian Pfeil. AERA Tests. https://github.com/shmax13/AERA_tests, 2026.

[24] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, second edition, 2003.

[25] Arash Sheikhlar. Autonomous causal generalization. 2024.

[26] Kristinn R Thórisson. From constructionist to constructivist ai. In *AAAI Fall Symposium: Biologically Inspired Cognitive Architectures*, pages 175–183, 2009.

[27] Kristinn R Thórisson. Empirical Reasoning Systems – Lecture Notes. http://cadia.ru.is/wiki/public:t-713-mers:mers-24:lecture_notes, 2024.

[28] Kristinn R Thórisson, Jordi Bieger, Xiang Li, and Pei Wang. Cumulative Learning. In *Artificial General Intelligence: 12th International Conference, AGI 2019, Shenzhen, China, August 6–9, 2019, Proceedings 12*, pages 198–208, 2019.

[29] Kristinn R. Thórisson, Jordi Bieger, Stephan Schiffel, and Deon Garrett. Towards Flexible Task Environments for Comprehensive Evaluation of Artificial Intelligent Systems and Automatic Learners. pages 187–196. 2015.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

[30] Kristinn R. Thórisson, Jordi Bieger, Thröstur Thorarensen, Jóna S. Sigurðardóttir, and Bas R. Steunebrink. Why Artificial Intelligence Needs a Task Theory — And What It Might Look Like. 5 2016.

[31] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A Standard Interface for Reinforcement Learning Environments. 11 2025.

[32] Alan M. Turing. Computing Machinery and Intelligence. In *Parsing the Turing Test*, pages 23–65. Springer Netherlands, Dordrecht, 2009.

[33] Pei Wang. On the working definition of intelligence. *Center for Research on Concepts and Cognition CRCC, Indiana University*, 1995.

[34] Pei Wang. Intelligence: From definition to design. In *International Workshop on Self-Supervised Learning*, pages 35–47. PMLR, 2022.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.