# HANDBOOK B — COURSE ENGINE DESIGN & RATIONALE

*A design record for capability-driven, governance-aware course infrastructure*

> **Canonical source:** This Markdown file.
> Word and PDF versions in `docs/handbook/` are derived artefacts and may be regenerated.

---

## STAGE 1 — PURPOSE, SCOPE, AND POSITIONING

*Why this handbook exists, and what it is for*

### 1. What This Handbook Is

This handbook documents the **design rationale** of the CloudPedagogy Course Engine.

Its purpose is to explain *why* the Course Engine was designed the way it was, *which principles informed its architecture*, and *what trade-offs were made* in translating capability and governance requirements into a working system.

Where the Course Engine Handbook (Handbook A) focuses on **how to use the tool responsibly**, this handbook focuses on **how and why the tool was designed**.

It is a design record, not a user guide.

### 2. What This Handbook Is Not

This handbook is **not**:

- a technical specification or API reference
- a replacement for code documentation or READMEs
- a tutorial on how to operate the Course Engine
- a marketing narrative or product pitch

Technical details belong in the repository documentation.
Operational guidance belongs in the Course Engine Handbook.

This document exists to capture **design intent**, **normative assumptions**, and **governance considerations** that cannot be inferred reliably from code alone.

## 3. Why a Design & Rationale Handbook Is Necessary

AI-enabled systems are often judged solely by what they do, rather than by the assumptions that shaped them.

Over time, this creates risk. Design decisions become implicit, boundaries are forgotten, and systems drift away from their original purpose — especially as contributors change or capabilities expand.

This handbook exists to reduce that risk.

By making design intent explicit, it supports:

- institutional review and assurance
- responsible future development
- informed critique and contribution
- continuity of purpose over time

It also provides a stable reference point for explaining why certain features exist — and why others do not.

## 4. Relationship to the Course Engine Handbook

This handbook is intended to be read **alongside**, not instead of, the Course Engine Handbook.

The relationship between the two is intentional:

- **Handbook A** explains how to use the Course Engine responsibly
- **Handbook B** explains why the Course Engine was designed the way it was

Handbook A is concerned with practice.
Handbook B is concerned with design reasoning.

Keeping these concerns separate avoids overloading users while still preserving a clear design record.

## 5. Relationship to the AI Capability Framework and CDD

The Course Engine is not a neutral technical artefact.

Its design is explicitly informed by:

- the **CloudPedagogy AI Capability Framework**, which defines what responsible AI capability looks like in educational and public-interest contexts
- **Capability-Driven Development (CDD)**, which provides a method for translating capability requirements into system design decisions

This handbook explains how those influences shaped:

- architectural choices
- defaults and constraints
- feature scope
- validation and reporting behaviour
- deliberate exclusions and non-features

It does not restate the full Framework or the full CDD method. Instead, it documents **how they were applied in this specific system**.

## 6. Intended Audience

This handbook is written for readers who need to understand the Course Engine at a **design and governance level**, including:

- institutional reviewers and assessors
- technical contributors and maintainers
- researchers examining AI-enabled educational infrastructure
- decision-makers evaluating adoption or alignment
- future contributors to the Course Engine itself

It assumes familiarity with professional, institutional, or system-level concerns, but it does not assume detailed knowledge of the codebase.

## 7. How to Read This Handbook

This handbook is designed to be read **selectively**.

Some readers may focus on early sections to understand overall positioning. Others may consult specific stages when reviewing a design decision or proposing a change.

It is not intended to be read linearly from start to finish, and it is not expected to change frequently. Stability of reasoning is treated as a governance concern.

## 8. Stability and Evolution

The Course Engine will evolve. This handbook exists to ensure that evolution remains **intentional**.

The principles and design commitments described here are intended to remain stable. However, new sections may be added to document additional design decisions, extensions, or clarifications as the system develops.

Changes to this handbook should favour **addition and explanation**, not retrospective rewriting.

### Stage 1 Summary

This stage establishes the purpose and scope of Handbook B.

It clarifies that this document exists to record design intent, explain architectural and normative choices, and support responsible evolution of the Course Engine over time.

The next stage moves from purpose to foundations, examining the **design problem the Course Engine set out to address** and the assumptions that shaped its initial conception.

---

## STAGE 2 — THE DESIGN PROBLEM THE COURSE ENGINE RESPONDS TO

*Why existing approaches to AI-supported course design were insufficient*

### 9. The Problem Is Not Content Production

The primary design problem the Course Engine responds to is **not** a lack of tools for producing educational content.

By the time this system was conceived, generative AI tools were already highly capable of drafting text, summarising material, and generating outlines. Speed and fluency were not the limiting factors. If anything, they were becoming a source of new risk.

The real problem lay elsewhere: **how AI was being introduced into course design without adequate structure, accountability, or governance**.

Courses could be produced quickly, but it was increasingly difficult to explain *why* particular decisions had been made, *who* was responsible for them, or *how* they could be reviewed and revised over time.

## 10. Fragmentation of Intent and Decision-Making

In many AI-supported workflows, intent is fragmented across tools and moments.

Design decisions may be partly embedded in prompts, partly implied by generated text, and partly held tacitly by individuals. Over time, this fragmentation makes it difficult to reconstruct the reasoning behind a course's structure, emphasis, or scope.

This becomes especially problematic when:

- courses are reviewed by others who were not involved in their creation
- responsibility needs to be demonstrated to an institution or regulator
- materials need to be updated months or years later

The design problem was therefore not how to generate better content, but how to **make intent explicit, durable, and reviewable**.

## 11. Automation Without Clear Boundaries

A second aspect of the problem was the **blurring of boundaries** between human judgement and automated assistance.

In many systems, AI output is presented as seamless and authoritative, with little indication of where human decision-making ends and automated generation begins. This can create the impression that decisions have been made "by the system," even when no such authority was intended.

From a governance and accountability perspective, this ambiguity is unacceptable. Responsibility cannot be meaningfully assigned if roles are unclear.

The Course Engine was designed in response to this problem: to ensure that **assistance never masquerades as authority**, and that human judgement remains visible rather than implicit.

## 12. Post-Hoc Governance and Retrofitted Assurance

A recurring pattern in AI-supported course design is the treatment of governance as an **afterthought**.

Courses are designed and published first, with questions about alignment, assurance, or capability mapping addressed later — often under time pressure and with limited evidence. This leads to retrofitting justifications onto artefacts that were never designed to support them.

The design problem here is structural. If courses are not designed from the outset to be inspected, reviewed, and explained, governance becomes reactive rather than embedded.

The Course Engine responds to this by treating **reviewability and auditability as design requirements**, not optional extras.

## 13. Loss of Coherence Over Time

Courses do not exist at a single moment.

They evolve in response to feedback, institutional change, and shifts in disciplinary or technological context. However, many AI-generated or AI-assisted courses are brittle. Once published, they are difficult to revise without effectively starting again.

This brittleness arises when structure is implicit, dependencies are unclear, and earlier decisions are not recorded. Over time, coherence erodes, and confidence in the material declines.

The design problem, therefore, was how to support **courses as long-lived artefacts**, capable of adaptation rather than replacement.

## 14. Over-Reliance on Tool-Centred Thinking

Another underlying issue was the tendency to frame course design problems in terms of tools rather than capabilities.

Questions such as "Which AI tool should we use?" or "What can this model generate?" often displaced more important questions about what educators and institutions needed to be able to do: explain decisions, maintain quality, support review, and adapt responsibly.

The Course Engine was designed to reverse this framing. Instead of starting with tools, it starts with **capability requirements** and asks what kind of system would be needed to support them.

## 15. The Absence of a Middle Layer

At a structural level, there was a missing **middle layer** between high-level frameworks and low-level tooling.

Frameworks could articulate principles and values, while tools could generate content. What was lacking was infrastructure that could translate those principles into **practical, inspectable design artefacts** without enforcing compliance or automating judgement.

The Course Engine was conceived to occupy this middle layer: not a framework, not a generator, but a system that could **hold intent, structure, and evidence together**.

From **v1.12**, this "hold intent together" requirement is supported through an optional `design_intent` block in `course.yml`. This provides a stable, inspectable place to record rationale, AI positioning, and governance context, and governance context as **informational metadata**, without turning intent into enforcement.

### 16. Summary of the Design Problem

Taken together, the design problem the Course Engine responds to can be summarised as follows:

AI made it easy to produce content, but difficult to:

- make intent explicit and durable
- preserve clear human–AI boundaries
- support governance and review without retrofitting
- maintain coherence over time
- design for capability rather than tooling

The Course Engine was not designed to solve all of these problems automatically. It was designed to **make them addressable by design**.

### Stage 2 Summary

This stage has described the problem space that motivated the Course Engine's design.

The challenge was not technical capability, but the absence of infrastructure that could support accountable, capability-aware, and governable course design in an AI-enabled environment.

The next stage examines the **foundational design principles** adopted in response to this problem, and how they shaped the Course Engine's overall architecture.

---

## STAGE 3 — FOUNDATIONAL DESIGN PRINCIPLES

*The commitments that shaped the Course Engine's architecture*

## 17. Principles as Design Constraints, Not Values Statements

The Course Engine was not designed by starting with a list of features. It was designed by committing to a small number of **foundational principles** and then treating those principles as *constraints* on what the system could and could not do.

These principles are not branding statements or abstract values. They function as **design tests**. When a proposed capability conflicted with a principle, the capability was revised, constrained, or rejected.

This stage documents those principles and explains how they were used to guide architectural and behavioural decisions throughout the system.

## 18. Human Judgement Must Remain Central

The first and most important principle is that **human judgement must remain central**.

The Course Engine assumes that decisions about educational purpose, scope, framing, and ethical appropriateness cannot be delegated to an automated system without loss of responsibility. As a result, the system is designed to assist with drafting, structuring, and inspection, but not to decide what should be taught or how it should be evaluated.

This principle led directly to design choices such as:

- avoiding automated course generation
- refusing to infer quality or effectiveness
- keeping AI assistance provisional and visible

Any feature that would obscure where judgement was exercised was treated as a risk rather than a benefit.

## 19. Structure Before Generation

A second foundational principle is that **structure should precede generation**.

Rather than allowing content to drive structure implicitly, the Course Engine requires that structure be defined explicitly before substantial text is produced. This reflects the view that coherence, progression, and interpretability depend on deliberate organisation, not on post-hoc editing.

This principle shaped the decision to treat the course specification as a first-class artefact and to separate structural design from content elaboration. It also influenced the system's resistance to "one-click" generation workflows that collapse design decisions into a single step.

## 19A. Intent Must Be Inspectable (v1.12+)

A recurring governance failure mode in AI-supported course design is that intent exists only in tacit knowledge or transient prompts.

From **v1.12**, the Course Engine supports an optional `design_intent` block to make rationale, AI positioning, and review context **explicit and inspectable**. This is a governance signal, not a compliance mechanism: it records author intent without asserting quality, correctness, or approval.

From v1.13, machine-readable AI boundaries can also be declared via `ai_scoping`, separate from narrative intent.

## 20. Transparency Over Seamlessness

Many AI-enabled systems prioritise seamlessness, hiding intermediate steps to create the impression of intelligence or autonomy.

The Course Engine adopts the opposite stance: **transparency is preferred to seamlessness**, even when this introduces friction.

This principle led to design choices such as:

- making build artefacts inspectable
- generating explicit manifests rather than opaque outputs
- exposing validation and reporting results rather than resolving them silently
- adding fail-fast guardrails for common authoring mistakes (v1.6)

The assumption is that friction can be productive when it supports understanding, review, and accountability.

## 21. Capability Before Tooling

Another foundational principle is that **capability requirements should drive system design**, not tool availability.

Rather than asking what existing AI tools could generate, the design process began by asking what educators and institutions needed to be able to do: explain decisions, demonstrate alignment, revise courses over time, and support governance without reducing judgement to compliance.

This principle aligns directly with a capability-driven approach to system design and explains why the Course Engine is framed as infrastructure rather than as an authoring tool.

## 22. Governance as a Design Requirement

Governance was treated as a **design requirement**, not a downstream concern.

The Course Engine was designed on the assumption that courses may need to be reviewed, audited, or explained to others who were not involved in their creation. This required that evidence, structure, and declared intent be available in a form that could support such review.

As a result, features such as metadata generation, reporting, and validation were designed to support inspection rather than enforcement. Governance is supported by making information available, not by automating approval.

In **v1.12**, this governance stance is extended to *rationale itself*: authors can record declared **design intent** (why the course is structured and positioned as it is, including AI boundaries) in a way that is inspectable and durable, without being evaluated as "correct".

## 23. Non-Destructive Defaults

Another important principle is that **the system should default to non-destructive behaviour**.

Course design is iterative, and responsible experimentation requires the ability to compare versions, revisit earlier decisions, and recover from mistakes. Defaults that overwrite or discard earlier artefacts undermine this process.

This principle shaped decisions such as:

- requiring explicit confirmation for destructive actions
- preserving build outputs by default
- avoiding irreversible transformations

The system is designed to support cautious progress rather than rapid but brittle iteration.

## 24. Separation of Concerns

The Course Engine deliberately separates concerns that are often collapsed in other systems.

Frameworks define norms and values.
Methods describe how design decisions are made.
Tools implement infrastructure.

The Course Engine positions itself strictly in the tooling layer. It supports frameworks and methods, but it does not embed them or enforce them. This separation allows each layer to evolve independently without conflating judgement with implementation.

### 25. Explicit Non-Goals

Finally, the design principles include a set of **explicit non-goals**.

The Course Engine was not designed to:

- replace pedagogical expertise
- automate curriculum approval
- score or rank courses
- infer educational quality
- act as an institutional authority

These non-goals were treated as important as the goals themselves. Features that moved the system toward these roles were deliberately excluded.

### 26. Principles as a Guard Against Drift

Taken together, these principles function as a guard against **design drift**.

As new capabilities are proposed or implemented, they can be evaluated against the same foundational commitments. This helps ensure that evolution remains intentional rather than incremental in unintended directions.

The purpose of documenting these principles is not to freeze the system, but to ensure that change remains **legible, contestable, and aligned**.

### Stage 3 Summary

This stage has articulated the foundational design principles that shaped the Course Engine.

These principles explain why the system behaves as it does, why certain features exist, and why others do not. They serve as constraints on implementation and as reference points for future development.

The next stage examines how these principles were translated into **specific architectural and behavioural choices** within the system.

---

## STAGE 4 — TRANSLATING PRINCIPLES INTO SYSTEM ARCHITECTURE

*How design commitments shaped concrete system decisions*

## 27. Architecture as an Expression of Design Intent

The architecture of the Course Engine is not accidental, nor is it merely a technical implementation of functional requirements.

It is an expression of the design principles outlined in the previous stage. Each major architectural choice reflects a deliberate attempt to preserve human judgement, maintain transparency, and support governance without enforcement.

This stage explains how those principles were translated into **structural and behavioural decisions** within the system.

## 28. Treating the Course Specification as a First-Class Artefact

One of the most significant architectural decisions was to treat the course specification (`course.yml`) as a **first-class artefact**, rather than as an internal configuration detail.

This reflects the principle of *structure before generation*. By making the specification explicit, inspectable, and durable, the system ensures that intent and structure are not hidden inside prompts or transient UI states.

The course specification becomes a shared object that can be reviewed, versioned, and discussed independently of any particular output. This supports collaboration, accountability, and long-term maintenance.

## 29. Separation Between Specification, Build, and Render Phases

The Course Engine deliberately separates the workflow into distinct phases: specification, build, and render.

This separation reflects multiple design principles simultaneously. It reinforces transparency by making intermediate artefacts visible, supports non-destructive defaults by allowing builds to exist independently of rendering, and enables inspection and reporting without forcing publication.

Architecturally, this avoids collapsing multiple decisions into a single opaque operation. Each phase can be understood, repeated, or revisited in isolation.

## 30. Explicit Manifest Generation for Inspection and Review

Rather than treating build metadata as an internal concern, the Course Engine generates an explicit, machine-readable manifest (`manifest.json`) as part of the build process.

This manifest exists to support inspection, reporting, and validation. It allows reviewers to understand what was produced, how it was structured, and what declarations were made, without needing to infer behaviour from outputs alone.

This design choice follows directly from the principles of transparency and governance-as-design. The manifest is not an optimisation; it is evidence.

## 30A. External Lesson Sources as an Architectural Extension (v1.6)

In v1.6, the Course Engine introduced support for authoring lessons as **external source files**.

This change did not alter the system's responsibility model or workflow assumptions. Instead, it represents an architectural extension that preserves existing design commitments:

- structure remains explicit and human-defined
- lesson inclusion remains declarative
- AI does not gain additional authority
- provenance and traceability are strengthened, not weakened

External lesson sources were treated as an authoring affordance, not as a mechanism for dynamic inclusion or automation. For this reason:

- lessons must still be explicitly declared in the course specification
- ambiguity between inline and external content is disallowed
- provenance is recorded as evidence, not optimisation

This extension demonstrates how the Course Engine can evolve authoring flexibility while preserving its foundational principles.

## 30B. Fail-Fast Authoring Guardrails (v1.6)

In v1.6, the Course Engine added **fail-fast preflight checks** for common authoring mistakes and ambiguous course layouts.

This is a governance-relevant architectural decision: it reduces "silent correctness" (where a build succeeds but the authored intent is misrepresented), and it produces clearer, actionable errors that keep responsibility visible.

This guardrail approach supports:

- faster diagnosis during authoring and review
- reduced ambiguity in how course specifications are interpreted
- stronger alignment with transparency over "helpful guesswork"

The system's bias is toward clear failure with explanation, rather than permissive behaviour that produces hard-to-trace outcomes.

## 30C. Design Intent as Manifest-Backed Metadata (v1.12)

In **v1.12**, the Course Engine introduced support for an optional `design_intent` block in `course.yml`.

This feature extends the manifest-as-evidence approach: it allows authors to record *rationale and boundaries* as structured, inspectable metadata, without requiring reviewers to infer intent from the rendered content alone.

Key architectural properties:

- **informational only**: intent is recorded, not enforced
- **manifest-backed**: intent is captured in `manifest.json` for auditability
- **change-detectable**: a stable hash is recorded to support provenance and drift detection
- **explain surfaced**: intent is visible via `course-engine explain` in both JSON and text forms

This preserves the system's core stance: governance is supported through transparency and evidence, not automated approval.

From **v1.13**, `ai_scoping` provides a **structural declaration** of **permitted**, **restricted**, and **expected** AI use. When present, it is recorded in `manifest.json` and can suppress advisory **absence signals** about missing AI boundaries.

### 30D. AI Scoping and Absence Signals (Informational, v1.13+)

`ai_scoping` is **informational, structural metadata** describing **permitted**, **restricted**, and **expected** AI use boundaries.

From v1.13, the Course Engine records **absence signals** in `manifest.json` to make **missing governance-relevant elements** explicit **without enforcement**.

Absence signals are **advisory by default**. They are designed to support **review, assurance, and governance conversations**, rather than to trigger compliance decisions or automated approval outcomes.

## 31. Informational Capability Mapping by Design

Capability mapping was implemented as **informational metadata**, not as an enforcement mechanism.

Architecturally, this meant designing capability declarations to be read, reported, and validated for presence and traceability, without allowing them to block builds or infer quality.

This reflects a deliberate refusal to collapse complex judgement into automated scoring. The system supports visibility and discussion, but it does not claim authority over what constitutes sufficient or appropriate capability development.

## 31A. Three Layers of Governance Signalling: Intent, Alignment, and Defensibility (v1.12)

The Course Engine supports **three distinct layers** of governance signalling:

1. **Design intent** (`design_intent`, v1.12+)
   A human declaration of rationale, AI positioning/boundaries, and review context. Informational only, captured for inspection and audit.
2. **Declared alignment** (`framework_alignment`, v1.6+)
   A human declaration that a course references a framework and domains. Useful for orientation, inventory, and high-level reporting.
3. **Defensible mapping evidence** (`capability_mapping`, v1.1+)
   Structured metadata (coverage and evidence) intended to support inspectable claims and stronger governance workflows.

In v1.6, reporting was extended so that if no `capability_mapping` exists but `framework_alignment` does, the system can still produce an informative summary — without implying evidence-based coverage.

Validation remains intentionally dependent on `capability_mapping`, reflecting the distinction between **intent**, **declaration**, and **defensibility**.

## 32. Validation as Explanation, Not Enforcement

Validation within the Course Engine was designed to support explanation rather than enforcement by default.

Architectural decisions here include:

- running validation against generated artefacts rather than during generation
- supporting non-strict modes that report issues without failing

- enabling strict modes only when explicitly requested

This ensures that validation supports assurance workflows without forcing compliance models onto all users.

## 33. Explain-Only Policy Resolution as a First-Class Mode

Explain-only policy resolution reflects a broader architectural commitment to transparency and separation of concerns.

By allowing policies and profiles to be resolved and inspected **without requiring a built course**, the system supports governance workflows, CI checks, dashboards, and institutional review processes where *policy interpretation must be visible* without producing new artefacts.

Architecturally, this required treating policy resolution as a distinct concern, rather than as a side effect of validation.

## 33A. Explainability as a Contract-Stable Governance Interface (v1.10+)

From v1.10 onwards, the Course Engine's explainability interface is treated as a **contract-stable governance surface**, not merely a debugging aid.

Explain outputs are explicitly designed to:

- surface resolved inputs, policies, profiles, and provenance
- support CI, QA, and institutional review workflows
- remain deterministic and machine-readable (timestamps aside)
- avoid making pedagogical, quality, or compliance claims

From **v1.12**, explain outputs may also surface declared **design intent** (when present) as a governance signal, while remaining descriptive rather than evaluative.

This interface is intentionally **separate from build artefacts** such as `manifest.json`. While the manifest records *what was built*, the explainability interface exists to explain **how decisions, rules, and resolution paths were derived**.

By stabilising explain output semantics and format selection, the Course Engine treats explainability as infrastructure for governance and audit — not as an implementation detail or transient debugging output.

## 34. Non-Destructive Defaults and Explicit Overrides

Non-destructive behaviour is enforced architecturally through defaults that preserve existing artefacts unless explicitly overridden.

This required conscious design decisions around file handling, output directories, and command semantics. The goal was to make destructive actions possible, but never implicit.

This approach aligns with the principle that iteration and experimentation should be safe, reversible, and auditable.

## 35. Minimal Assumptions About Downstream Platforms

The Course Engine avoids embedding assumptions about where or how outputs will be consumed.

Rather than coupling the system tightly to a particular learning platform or delivery environment, it produces portable artefacts that can be rendered, hosted, or integrated elsewhere.

Architecturally, this reinforces separation of concerns and avoids locking design decisions into platform-specific constraints.

## 36. Guarding Against Feature Creep

Finally, the architecture reflects an intentional resistance to feature creep.

Capabilities were added only where they supported the system's core purpose: making course design more explicit, reviewable, and governable. Features that would blur responsibility, automate judgement, or obscure intent were treated as architectural risks.

This discipline is visible in what the system does *not* include as much as in what it does.

## Stage 4 Summary

This stage has shown how foundational principles were translated into architectural decisions within the Course Engine.

The system's structure is not incidental. It is a direct expression of commitments to human judgement, transparency, capability awareness, and responsible evolution.

The next stage examines the **deliberate exclusions and trade-offs** that accompanied these decisions, and why certain tempting capabilities were intentionally left out.

---

## STAGE 5 — DESIGN TRADE-OFFS AND DELIBERATE EXCLUSIONS

*What the Course Engine chose not to do — and why*

### 37. The Importance of Explicit Non-Features

Every system embodies trade-offs, whether or not they are acknowledged.

In the design of the Course Engine, certain capabilities were consciously **excluded**, not because they were technically infeasible, but because they conflicted with foundational principles. Making these exclusions explicit is essential for governance, critique, and future development.

This stage documents the most significant non-features and the reasoning behind them.

### 38. No Automated Course or Curriculum Generation

One of the most visible exclusions is the absence of automated, end-to-end course generation.

The Course Engine does not offer a workflow in which a user describes a topic and receives a completed course in response. While such functionality is common elsewhere, it was rejected because it collapses multiple layers of judgement into a single automated act.

Automated generation risks:

- obscuring where decisions are made
- encouraging acceptance of content without review
- undermining accountability for structure and scope

The Course Engine instead supports *assisted drafting within explicit structure*, preserving human ownership of design decisions.

### 39. No Inference of Pedagogical Quality or Effectiveness

The system does not attempt to evaluate whether a course is pedagogically sound, effective, or appropriate for a given audience.

While metrics, heuristics, or models could be used to approximate such judgements, doing so would create a false sense of authority. Pedagogical quality is contextual, contested, and value-laden.

By refusing to infer quality, the Course Engine avoids presenting evaluative claims that it cannot responsibly justify. Judgement remains with educators, reviewers, and institutions.

## 40. No Automated Approval or Compliance Decisions

Another deliberate exclusion is automated approval.

The Course Engine supports inspection, reporting, and validation, but it does not approve courses, certify alignment, or determine compliance. Even in strict validation modes, the system reports unmet rules rather than declaring outcomes as acceptable or unacceptable in absolute terms.

This distinction is critical. Approval is an institutional and professional act, not a technical one. Automating it would misrepresent responsibility and authority.

## 41. No Implicit Enforcement of Frameworks

Although the Course Engine supports capability mapping and validation, it does not embed or enforce any specific framework by default.

Frameworks define norms and values. Tools implement infrastructure. Collapsing these roles risks turning frameworks into rigid compliance mechanisms and tools into unaccountable authorities.

By keeping frameworks external and declarative, the Course Engine allows users and institutions to apply their own interpretive judgement rather than inheriting assumptions embedded in software.

## 42. No Hidden Optimisation or Ranking

The system does not rank courses, optimise for engagement metrics, or score alignment.

While such features might appear attractive, they introduce competitive and instrumental logics that are poorly suited to reflective design and governance contexts. They also tend to obscure the basis on which judgements are made.

The Course Engine prioritises explainability and reviewability over optimisation.

## 43. No Mandatory AI Usage

AI assistance is not mandatory within the Course Engine workflow.

The system is designed so that courses can be specified, built, and reviewed without invoking generative AI at all. This ensures that the infrastructure remains usable in contexts where AI use is restricted, contested, or deliberately limited.

This exclusion reinforces the principle that AI is an assistant, not a prerequisite.

### 44. Trade-Offs Accepted by These Exclusions

These exclusions come with acknowledged trade-offs.

The Course Engine is:

- slower than one-click generation tools
- less immediately impressive in demonstrations
- more demanding of user engagement and judgement

These trade-offs were accepted deliberately. The design prioritises long-term responsibility, defensibility, and adaptability over short-term efficiency or novelty.

### 45. Exclusions as a Guard Against Mission Creep

Documenting exclusions serves a second purpose: guarding against mission creep.

As the system evolves, there will be ongoing pressure to add features that promise convenience, automation, or optimisation. This stage provides a reference against which such proposals can be evaluated.

Exclusions are not permanent prohibitions, but they are not accidental gaps either. Any future change that revisits them should do so explicitly and with clear justification.

### Stage 5 Summary

This stage has outlined the most significant design trade-offs and deliberate exclusions in the Course Engine.

By refusing to automate judgement, enforce compliance, or infer quality, the system preserves clarity about responsibility and authority. These non-features are essential to the integrity of the overall design.

The next stage examines how the Course Engine positions itself **within the broader CloudPedagogy ecosystem** and how its design relates to frameworks and methods without embedding them.

---

## STAGE 6 — POSITIONING THE COURSE ENGINE WITHIN THE CLOUDPEDAGOGY ECOSYSTEM

*How the system relates to frameworks, methods, and other tools*

## 46. The Course Engine as Infrastructure, Not Authority

Within the CloudPedagogy ecosystem, the Course Engine is intentionally positioned as **infrastructure**.

It does not define norms, values, or expectations. It does not decide what responsible practice looks like. Instead, it provides a structured environment in which such decisions can be expressed, inspected, and revised.

This positioning matters. By remaining infrastructural, the Course Engine avoids claiming authority that properly belongs to educators, institutions, and frameworks.

## 47. Relationship to the AI Capability Framework

The CloudPedagogy AI Capability Framework defines **what responsible AI capability looks like** across a set of domains.

The Course Engine does not reinterpret or restate the Framework. Instead, it provides mechanisms through which courses can:

- declare alignment
- document intent
- expose coverage and gaps (when mapping evidence exists)
- support review and assurance

The framework remains normative and interpretive. The Course Engine remains procedural and evidential.

This separation allows the framework to evolve conceptually without forcing changes in tooling, and allows the tooling to evolve without redefining capability norms.

## 48. Relationship to Capability-Driven Development (CDD)

Capability-Driven Development provides a method for translating capability requirements into system design decisions.

The Course Engine is not an implementation of CDD as a general method. It is, however, an applied example of **CDD in practice**.

Its architecture reflects a capability-first approach:

- intent is specified before automation
- human–AI boundaries are explicit
- governance and review are designed in, not bolted on

This handbook exists partly to make that application explicit, so that the Course Engine can be examined as a concrete instantiation of CDD principles without being mistaken for the method itself.

## 49. Relationship to Other CloudPedagogy Tools

The Course Engine is one component within a broader set of CloudPedagogy resources, which may include diagnostic tools, reflective instruments, and applied learning materials.

It is designed to interoperate conceptually with these resources, but not to depend on them. Outputs from the Course Engine can be informed by diagnostic or reflective work, and its artefacts may feed into other review or planning processes.

However, the Course Engine does not assume the presence of any particular upstream or downstream tool. This preserves modularity and reduces coupling across the ecosystem.

## 50. Avoiding Framework Lock-In

A key design concern was avoiding **framework lock-in**.

While the Course Engine was developed in close alignment with the CloudPedagogy AI Capability Framework, it was deliberately designed to support alternative or additional frameworks through declarative metadata.

This ensures that institutions can adapt the tool to their own contexts without inheriting implicit commitments embedded in software. Frameworks remain choices, not dependencies.

## 51. Supporting Multiple Levels of Use

The Course Engine is designed to support use at multiple levels simultaneously:

- individual educators drafting materials
- teams collaborating on course design
- institutions reviewing or assuring provision

Its positioning within the ecosystem reflects this multiplicity. It does not privilege any single level of use, nor does it enforce a particular workflow.

This flexibility is a design outcome, not an accident.

### 52. The Course Engine as a Boundary Object

In practice, the Course Engine functions as a **boundary object**.

It provides shared artefacts — specifications, manifests, outputs — that different stakeholders can engage with for different purposes. Educators, designers, reviewers, and technologists can all interact with the same artefacts without needing to agree on every underlying interpretation.

This role is particularly important in governance contexts, where shared reference points enable discussion without forcing consensus.

### 53. Positioning as an Enabler, Not a Solution

Finally, the Course Engine is positioned as an **enabler**, not a solution.

It does not claim to solve the challenges of AI-supported education. Instead, it aims to make those challenges more visible, more discussable, and more manageable through design.

This modest positioning is intentional. It reflects an understanding that responsibility cannot be automated, but it can be better supported.

### Stage 6 Summary

This stage has clarified how the Course Engine fits within the CloudPedagogy ecosystem.

By positioning itself as infrastructure, not authority, and by maintaining clear boundaries between frameworks, methods, and tools, the Course Engine supports responsible practice without overreach.

The final stage draws these threads together and reflects on how this design record should be used going forward.

---

## STAGE 7 — USING THIS DESIGN RECORD GOING FORWARD

*How this handbook should inform use, review, and evolution*

## 54. This Handbook as a Design Record

This handbook is intended to function as a **design record**.

It captures the reasoning, assumptions, and constraints that shaped the Course Engine at a particular point in its evolution. Its purpose is not to prescribe behaviour, but to make the system's design intelligible to others.

As such, it should be read as an explanation of *why things are the way they are*, not as a guarantee that they will never change.

## 55. Supporting Responsible Evolution

The Course Engine is expected to evolve.

New features, workflows, and integrations may be added over time. This handbook exists to ensure that such changes remain **intentional rather than incidental**.

When proposing or reviewing changes, this design record can be used to ask:

- Does this change preserve clear human–AI boundaries?
- Does it support capability development rather than tool-centred optimisation?
- Does it strengthen or weaken transparency and reviewability?
- Does it shift responsibility in ways that need to be made explicit?
- Does it blur the distinction between declared alignment and defensible mapping evidence?

These questions are more important than any individual feature request.

## 56. Guiding Contribution and Review

For contributors, this handbook provides context that cannot be inferred from code alone.

It explains why certain patterns recur, why certain shortcuts were avoided, and why some apparently attractive capabilities were deliberately excluded. Understanding this context helps contributors work *with* the system's intent rather than against it.

For reviewers and assessors, the handbook provides a basis for informed critique. Disagreement is expected and legitimate, but it should be grounded in an understanding of the original design commitments.

## 57. Preventing Unintentional Drift

Over time, systems tend to drift.

As pressures change and new possibilities emerge, incremental decisions can accumulate into significant departures from original purpose. This handbook is one mechanism for resisting unintentional drift.

By returning to the documented principles and trade-offs, maintainers can distinguish between:

- intentional evolution
- necessary adaptation
- accidental erosion of boundaries

Drift is not prevented by freezing a system, but by making change legible.

## 58. Relationship to Other Documentation

This handbook does not replace other forms of documentation.

Technical behaviour is documented in READMEs and code comments.
Operational guidance is provided in the Course Engine Handbook.
Change history is tracked through versioning and changelogs.

This document sits alongside those artefacts, addressing questions they cannot easily answer: questions of intent, rationale, and responsibility.

## 59. Stability as a Governance Concern

The relative stability of this handbook is intentional.

While it may be extended over time, core sections should not be rewritten lightly. Changes to the underlying design rationale represent governance-relevant decisions and should be treated accordingly.

This does not mean the design is fixed. It means that changes should be acknowledged, explained, and documented rather than silently absorbed.

## 60. Closing Reflection

The Course Engine was designed in response to a specific set of concerns about AI-supported course design: concerns about accountability, governance, and the erosion of judgement.

This handbook records one response to those concerns. It does not claim to be the only possible response, nor the final one.

Its value lies in making the reasoning explicit — so that responsibility remains visible, critique remains possible, and evolution remains intentional.

## Final Summary

This design and rationale handbook has documented:

- the problem space the Course Engine responds to
- the principles that shaped its design
- the architectural decisions that followed (including v1.6 extensions)
- the trade-offs and exclusions that were accepted
- the system's positioning within a broader ecosystem

Together, these sections provide a durable reference for understanding, using, and evolving the Course Engine responsibly.

---

## Relationship to the Course Engine Handbook (Handbook A)

This handbook documents the **design intent and rationale** of the CloudPedagogy Course Engine.

It does not provide operational guidance on how to use the system in practice.

For practical guidance on responsible use — including workflow expectations, human–AI boundaries, risks, and limits — see:

**Handbook A — Course Engine Handbook**

That handbook is written for educators, learning designers, technologists, and institutions who are using the Course Engine and remain accountable for the outcomes of that use.

The two handbooks are complementary:

- this handbook explains *why the system is designed as it is*
- the Course Engine Handbook explains *how to use it responsibly*