A Collaborative System for Structured Design Exploration: C-K Git

---

Capstone Project Report

Team Members:

Maggie Qin

Milan Liessens Dujardin

Yijing (Cindy) Ling

Advisors:

Kevin Ma

Kosa Goucher-Lambert

University of California, Berkeley

2026

Contents

Executive Summary **2**

1 Problem Space and Approach **3**

1.1 Introduction 3

1.2 Collaborative Design 3

1.3 C-K Theory 4

1.4 C-K Git 7

1.5 Contributions 8

2 C-K Git **10**

2.1 C-K Components 10

2.2 Single-Designer Interaction 12

2.3 Multiple-Designer Interaction 13

2.4 Large Language Model Operations 14

2.5 Usage Scenario 17

2.6 Key Insights 18

2.7 Limitations 19

2.8 Future Directions 19

2.9 Conclusion 20

Executive Summary

This project explores how artificial intelligence can support collaborative design within functions and areas of engineering and design during the early stages of innovation. While large language models (LLMs) are effective at generating ideas, most current AI tools function as isolated conversational systems that do not preserve reasoning, connect ideas to supporting knowledge, or support collaboration between multiple contributors.

To address this gap, we developed C-K Git, a collaborative design reasoning system inspired by Concept–Knowledge (C-K) Theory and version-control workflows such as Git. C-K Theory distinguishes between concept space, which contains emerging and uncertain ideas, and knowledge space, which contains validated information and assumptions. Design progresses through interaction between these spaces, allowing ideas to evolve through reasoning and refinement.

C-K Git provides a structured workspace where users can explore concepts, connect them to knowledge, and collaborate through branching and merging workflows. Designers can independently develop ideas and later merge their work into a shared concept space while preserving ownership and reasoning history.

Rather than replacing human creativity, the system supports transparent collaboration by making design reasoning visible and traceable within product, user experience, or engineering design. This project demonstrates how AI can move beyond idea generation toward supporting collaborative exploration and structured decision-making in design.

1 Problem Space and Approach

1.1 Introduction

Design is rarely an individual activity. In fields such as product development, user experience design, engineering, and innovation strategy, ideas are often developed collaboratively among multiple contributors. Designers, engineers, researchers, and stakeholders each bring different perspectives, assumptions, and expertise to the design process. Collaboration allows teams to generate diverse ideas, but it also introduces complexity, especially during the early stages of design.

Early-stage design differs from later implementation phases because solutions are not yet defined. Teams are not selecting from finalized options; instead, they are exploring possibilities. Ideas remain incomplete, uncertain, and subject to change. A design challenge may begin with a broad goal such as improving a user onboarding experience or creating a more accessible product, but the path toward a solution is rarely linear.

At the same time, advances in artificial intelligence have introduced new tools for creative support. Large language models (LLMs) can generate ideas, summarize information, and suggest alternatives quickly. However, most AI systems today function as conversational assistants. They produce outputs efficiently, but they do not naturally preserve how ideas evolve, how reasoning develops, or how multiple contributors collaborate around shared concepts.

This project investigates how AI can move beyond idea generation to support collaborative design reasoning. We explore how ideas, knowledge, and collaboration can be organized within a shared system.

1.2 Collaborative Design

The early stages of design are often difficult to structure because ideas emerge through exploration rather than clear decision-making. Designers must continuously move between generating possibilities, evaluating assumptions, and incorporating knowledge from multiple sources.

One challenge is that ideas often become disconnected from their underlying reasoning. Teams may remember the final outcome of a brainstorming session, but lose track of why a particular idea was proposed or what evidence supported it. As ideas accumulate, it becomes difficult to revisit earlier thinking or compare competing directions.

For example, consider a team designing a mobile onboarding experience. One designer may propose a guided tutorial to explain features step-by-step, while another suggests an adaptive onboarding flow that responds to user behavior. Both ideas may be valuable, but they reflect different assumptions about how users learn as well as their preferences. Without preserving the reasoning behind these proposals, teams may struggle to understand how the ideas differ or which direction should be explored further.

A second challenge is the weak connection between ideas and knowledge. Design decisions are often influenced by research findings, technical constraints, industry practices, or user insights. However, traditional brainstorming methods rarely connect concepts directly to supporting knowledge. As a result, designers may generate ideas that conflict with existing but unknown information or overlook opportunities.

Collaboration introduces additional complexity. Multiple contributors often explore the same design problem in parallel, producing different branches of thought. One designer may focus on usability, another on business goals, and another on technical feasibility. While this diversity is valuable, current tools provide limited support for merging these parallel explorations into a coherent shared understanding.

Although generative AI can rapidly produce ideas, most systems still operate as isolated exchanges between a user and a model. They rarely preserve collaboration history, track reasoning trajectories, or help reconcile conflicting contributions. This creates a gap between idea generation and collaborative design reasoning.

1.3 C-K Theory

Our project is inspired by Concept–Knowledge Theory (C-K Theory), a framework developed in design research to explain how innovation emerges through structured reasoning.

C-K Theory separates design thinking into two interacting spaces:

- **Concept Space (`C`):** ideas that are uncertain, exploratory, or not yet validated.
- **Knowledge Space (`K`):** information that is accepted, supported, or logically established.

`K`, the knowledge space, consists of validated information about the world. This includes established facts, experimental results, user feedback, and prior experiences. It represents what is known to the designers.

C, the concept space, in contrast, contains ideas. These are hypothetical solutions for a given design goal.

The design goal defines the target problem or objective of the exploration. It is expressed as an initial concept, which serves as the starting point of the exploration. The initial concept is to be iteratively refined, evaluated against knowledge in **K**, and progressively transformed into a validated solution within **C**.

C-K Theory defines the design process as the interplay between these two spaces: knowledge guides concept refinement, and concepts may demand additional knowledge for them to be validated as solutions to the design goal. Designers may also reason over existing knowledge or add new findings directly, expanding **K**, or generate creative solutions, adding concepts. These four possible interactions form the four C-K operations.

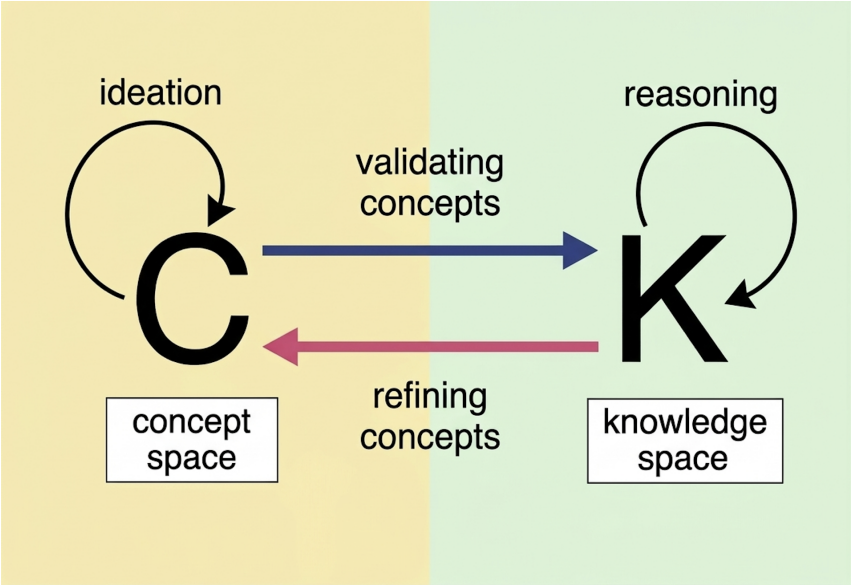

Figure 1: Concept–Knowledge (C-K) Theory framework illustrating interactions between concept space and knowledge space: validating concepts requires knowledge, and knowledge can help refine concepts. Create concept generation happens within **C**. Designers may reason over existing knowledge in **K**, adding knowledge. Each of the arrows in the figure represents an operation.

A simple example illustrates this relationship. Suppose a team wants to design an underwater diver communication mask that allows divers to communicate more clearly and comfortably during extended underwater operations—the design goal. A designer has populated the knowledge space with existing knowledge.

Knowledge Space (K):

- Verbal communication underwater is often unclear or unreliable.
- Divers may need to communicate in low-visibility and high-stress environments.
- Masks must remain sealed and comfortable during long periods of use.

Concept Space (C):

- An underwater diver communication mask that improves communication and comfort

These concepts begin as uncertain possibilities. As new knowledge becomes available, concepts can evolve. For example, the following facts are added to the knowledge space:

- New **knowledge:** Full-face masks can support integrated audio systems more effectively than standard mouthpiece setups.
- New **knowledge:** Noise reduction technology can improve voice clarity in underwater environments.

With this new information the designer can refine the initial concept, generating a more refined concept and potential solution:

- New **concept:** A full-face communication mask with built-in noise reduction for clearer underwater speech.

As the design process evolves, the designer will work their way to a concrete solution that matches the design requirements—an underwater diver communication mask that improves communication and comfort during underwater operations.

As highlighted by the example and 1, the interaction between **C** and **K** creates a structured reasoning process. Instead of treating design as a collection of isolated ideas, C-K Theory emphasizes how ideas emerge, evolve, and become refined through knowledge.

Another key benefit of C-K Theory is that it is domain-agnostic: it is applicable to any type or area of design, making it a universal framework suitable to support our collaborative system.

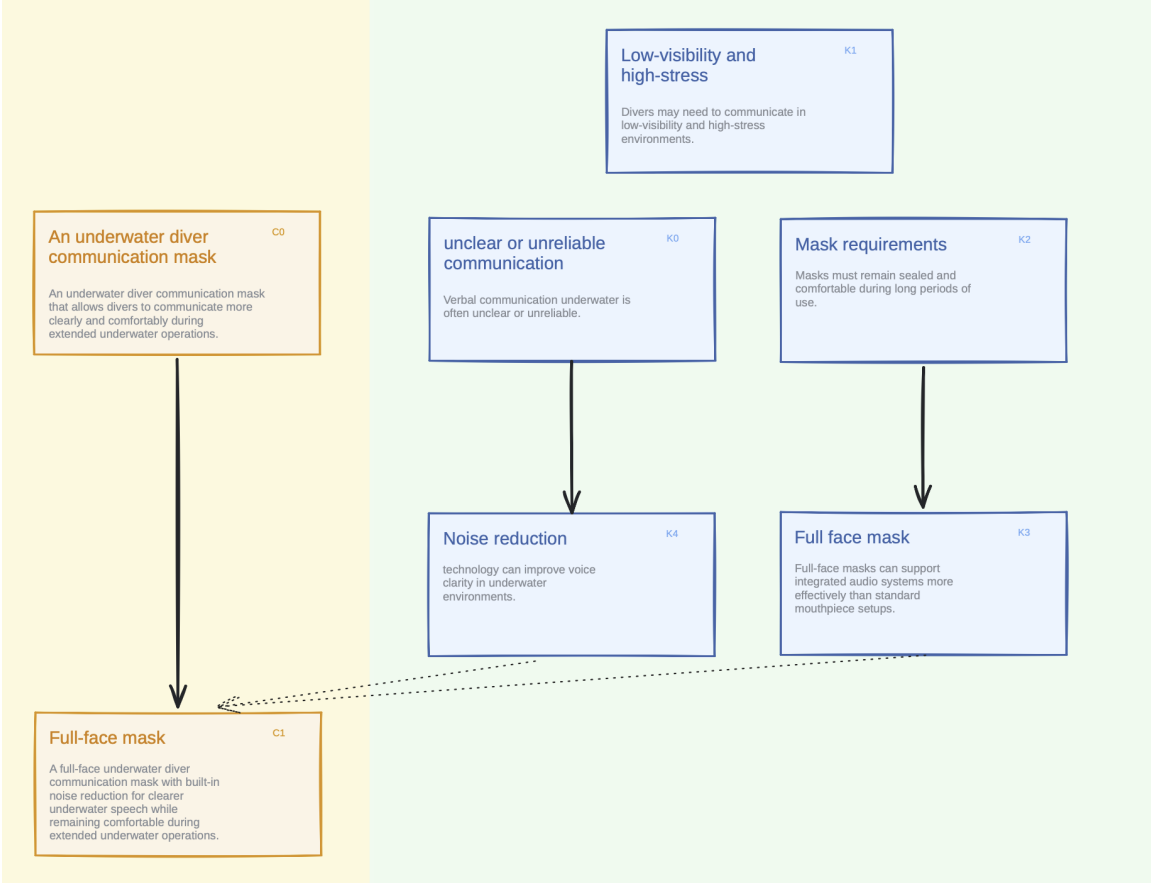

Figure 2: C-K Git example in the interface.

1.4 C-K Git

To address the challenge of structured design collaboration, we developed C-K Git, a collaborative design reasoning system that combines C-K Theory with Large Language Models (LLMs).

C-K Git acts as a shared workspace for collaborative exploration. The system allows users to develop ideas, connect them to knowledge, and preserve the reasoning behind design decisions. LLMs support designers with generating additional knowledge, validating concepts, and merging ideas.

The name "C-K Git" reflects inspiration from version-control systems such as Git. In software development, Git allows multiple contributors to work independently, preserve history, and later merge their work. Our project applies a similar structure to conceptual design.

Users begin with an identical initial concept and explore different directions. Multiple contributors

can work in parallel, each expanding ideas based on their own perspective.

For example, a team exploring onboarding design may produce several branches:

- One branch focused on personalization.
- One branch focused on accessibility.
- One branch focused on reducing cognitive load.

Rather than forcing these ideas into a single brainstorming session, the system preserves the different explorations as parallel tracks and supports their convergence when teams come together.

LLMs support this process by helping users generate concepts, expand knowledge, and surface supporting information. AI may introduce relevant research insights, identify assumptions, or suggest related design directions. Importantly, the system does not simply produce ideas; it records the relationships between concepts and knowledge.

Beyond the interaction of a single user with the system, a key contribution of the platform is its merge workflow. When multiple users explore the same concept independently, the system compares and contrasts their concept and knowledge spaces and brings them into a shared structure. During the merge process, conflicting knowledge and concepts are identified and fed back to the contributors. This allows for effective conflict resolution, helping designers reconcile ideas collaboratively. Because the merge is designed to preserve contributor ownership, reasoning history, and supporting knowledge, it empowers human decision-making in design as opposed to replacing it.

1.5 Contributions

The goal of C-K Git is not only to help designer navigate the design space creatively and in a structured manner, but also to support a more structured form of collaboration during early-stage design.

Concretely, our contributions are the following:

- First, C-K Git supports **collaborative exploration** by allowing multiple contributors to work in parallel. By design, the system encourages branching reasoning paths.
- Second, C-K Git improves **transparency**. Design decisions often become difficult to explain once ideas evolve over time. By preserving relationships between concepts and knowledge and the reasoning trajectory, the system makes the design process visible and traceable.

- Third, our system eases **validation** in collaborative settings. This helps teams evaluate concepts more thoughtfully rather than treating creativity and evidence as separate activities.
- Finally, our system is fundamentally **human-centered**. AI supports reasoning and exploration, but final judgment remains with the designers.

By combining collaborative workflows, AI reasoning support, and C-K Theory, C-K Git demonstrates how AI can move beyond idea generation toward supporting shared exploration and structured design collaboration.

2 C-K Git

C-K Git is a collaborative system for structured design exploration that employs C-K Theory as its guiding framework and leverages language models to operationalize it. As discussed in the previous section, C-K Theory has affordances that make it a suitable framework to support structured collaborative design exploration.

C-K Git supports both single-designer and multiple-designer workflows. In single-designer scenarios, a designer interacts with the system by iteratively expanding the concept and knowledge spaces until they reach a satisfactory solution to the design problem. In collaborative settings, designers collectively define the design goal and then interact with the system similarly as in the single-designer setting—until they agree to reconcile their explorations. This is the point in the workflow where concepts are compared and contrasted and eventually merged. The resulting state or the system becomes the starting point for the next iteration.

In this section, we discuss our system in detail. First, we dive deeper into C-K Theory and its application within our system. Then, we outline the interaction between a single designer and system as well as multiple designers. We discuss the role and responsibilities of LLMs as the agents acting on behalf of the theory and supporting the design process. We present the principles that underlie the system and concretize it with a scenario. We end our discussion with the limitations of our work and future directions.

2.1 C-K Components

C-K Git implements C-K Theory as the framework that guides the exploration process. In this section, we discuss our representation of the Concept and Knowledge space, as well as the implementation of the four C-K operations.

2.1.1 C-Space Representation

The `C` space is where designers iterate on concepts to find a solution to the design goal or problem. The design goal, represented as the initial concept, sets the design requirements or initial idea. Every subsequent concept is a refinement of this initial concept—a hypothetical solution to the goal. As a result, the concept space is a tree of which every new concept is derived from its root, the initial concept. Each concept is defined by its attributes and parameters.

Our practical implementation slightly deviates from the original formulation of C-K Theory in one deliberate respect: rather than defining concepts as propositions with no logical status until they are validated or refuted by knowledge (i.e. ideas whose existence as a potential solution is undecided), we let every concept be in one of three states: approved, rejected, or undecided. A concept is approved when the designer accepts it as a potential solution given the current state of **K**, rejected when the designer discards it, and undecided whenever it is still being explored, is incomplete, or cannot yet be settled on the available evidence. As a result, the concepts always remain in the Concept space, even after gaining a logical status.

2.1.2 K-Space Representation

The knowledge space contains all validated information relevant to the design problem, including domain knowledge, constraints, empirical insights, and experiments. The knowledge can be added by the designer with or without LLM assistance, derived from existing knowledge or inspired by existing concepts. Because of this, **K** is structured as an archipelago: new knowledge either expands or connects existing islands, or creates new ones.

2.1.3 C-K Operations

In C-K Theory, four operations drive the expansion of the two spaces: **C** $\rightarrow$ **K**, **K** $\rightarrow$ **K**, **K** $\rightarrow$ **C**, and **C** $\rightarrow$ **C**. Our system supports them in the following ways:

C $\rightarrow$ K: reasoning over and validating concepts

This operation involves adding new knowledge related to an existing concept. This can be by running an experiment testing the concept, or any new information produced based on the concept itself, its attributes or parameters. This operation also includes determining whether a new concept in the **C** space is approved or rejected given current **K**.

K $\rightarrow$ C: ideation using knowledge

This operation includes refining concepts by adding or changing attributes based on knowledge in the **K** space, as well as proposing new concepts.

K $\rightarrow$ K: reasoning

This operation involves any type of reasoning, including classification, deduction, abduction, and inference, over existing **K**. Designers may also insert entirely new, unconnected knowledge.

C → C: creative ideation

This part involves creating new concepts or iterating on existing ones using creative thinking: it does not necessarily derive from existing knowledge.

2.2 Single-Designer Interaction

C-K Theory defines design as the interplay between the knowledge and concept spaces. Starting from the initial concept, the designer alternates between reasoning over **K**, expanding **K** with new information, refining concepts with **K**, and validating the resulting concepts against **K**. Each iteration either closes a branch, yields a candidate solution, or produces a new undecidable concept that becomes the focus of the next iteration.

Starting from a given concept, a designer may execute the following steps:

1. The designer reasons over the concept and adds knowledge related to it (**C** → **K**).
2. The designer adds knowledge unrelated to that specific concept but related to the project (**K** → **K**).
3. The designer refines the concept into several new concepts, "sub-concepts", based on attributes declared important by **K** (**K** → **C**).
4. The designer validates the sub-concepts against **K** (**C** → K).

The validation step can produce three outcomes:

- If the sub-concept is approved as a potential solution, the designer assesses whether it is fully specified — that is, whether its attributes and parameters are defined with sufficient precision. If so, the design process may conclude.
- If the sub-concept is rejected, the designer continues the design process, starting from an approved—but not yet complete—or undecidable concept.
- If the sub-concept is undecidable, the designer continues their exploration.

As Hatchuel and Weil said, "Undecidability and incomplete concepts can be seen as consistent triggers" (Hatchuel and Weil, 2009): the designer repeats this process of applying C-K operations for as long as the designer does not reach a satisfactory solution or abandons the process.

2.3 Multiple-Designer Interaction

Human collaboration hinges on common ground and shared semantics. Yet unifying divergent exploration tracks is inherently challenging — collaborators may have little visibility into one another's reasoning. C-K's formulation of design as a structured, iterative process makes it uniquely suited to address this problem: because every design move is traceable to an explicit concept refinement or operator application, divergent explorations of the same initial concept can be systematically compared and conflicts resolved. Knowledge and concept spaces from multiple designers can be contrasted or selectively combined, and — when convergence is desired — concept spaces can be merged. The merge operation unifies the work of two or more designers operating on the same initial concept, and is defined as follows.

2.3.1 Merge

After exploring the design space individually, designers may want to compare and contrast their ideas in order to converge toward a shared solution. This is enabled by the merge operation, which takes separate **`C`** and **`K`** spaces and surfaces their differences to the collaborating designers. In the language of the double diamond, the merge executes the convergence phase after a divergence phase: divergence is human-led and LLM-assisted, while convergence is LLM-led and human-controlled.

On the **`K`** side, the merge does not unify knowledge spaces — each designer retains their own, which may be updated or refined in light of what collaborators know, but remains individually owned. The merge flags conflicting propositions arising from K-relativity (the fact that each designer has been reasoning inside their own local truth), identifies propositions that are semantically equivalent across spaces, and reports every discrepancy back to the designers for resolution. Reorganization of one's **`K`** space in response to these contrasts is left to the designer's discretion.

On the **`C`** side, the merge constructs a single valid concept tree from the designers' separate trees. Unlike the **`K`** side, the concept space is genuinely merged: attributes are integrated hierarchically, making this fundamentally a tree-restructure operation rather than a concatenation. Where the same concept appears in both trees with different states — approved, rejected, or undecided — the merge surfaces the disagreement, leaving resolution to the designers.

Three principles discipline the operation:

1. The merge reorganizes, contrasts, and recombines information. It never deletes any.

2. The merge tracks user ownership, meaning who designed what, so that after reconciliation every entry can still be traced to its author.
3. The merge defers every substantive conflict-resolution decision to the users.

Concretely, the merge executes in six steps:

1. **Knowledge conflict detection.** The system compares `K` entries across designers, identifying pairwise clashes as well as contradictions that emerge only in combination — including statements that are individually plausible but jointly inconsistent, or propositions that are factually false. Detected conflicts are stored for user review.
2. **Knowledge conflict resolution.** The system presents each conflict in turn, explains the source of the disagreement, and prompts the designers to resolve it: retain one side, the other, or formulate a new proposition. Resolution is always left to the humans.
3. **Knowledge-space contrast and refinement.** With conflicts resolved, the system identifies semantically equivalent propositions across the two `K` spaces, surfaces complementary knowledge that may enrich either designer's understanding, and reports all changes. Each designer's `K` space remains their own — updates are suggested, not imposed.
4. **Validation propagation.** The system re-evaluates every concept against each designer's updated `K` space, rechecking whether its state — approved, rejected, or undecided — still holds.
5. **Concept-level conflict detection.** The system identifies concepts that appear in both trees and flags those whose states diverge between the two designers.
6. **Concept tree merge and conflict resolution.** The system presents concept-level conflicts to the designers for resolution, then constructs a single merged concept tree — integrating attributes hierarchically and preserving the partition structure — once all disagreements have been addressed.

2.4 Large Language Model Operations

We leverage the generative power, vast knowledge, and language understanding of large language models to support both the individual and collaborative design process. Given the abstract nature and complexity of C-K Theory, LLMs are crucial to ensuring its structure is applied consistently throughout the designer-system interaction. They serve two purposes: supporting the design process itself, and operationalizing C-K Theory as a collaborative guiding framework.

2.4.1 Supporting the Design Process

LLMs retrieve knowledge, track reasoning history, and validate concepts. They also flag knowledge conflicts, support merging, and suggest new avenues of exploration. Their ability to surface knowledge gaps, propose new concept attributes, synthesize related knowledge, and restructure spaces is what makes them valuable within the system. These capabilities are exposed to the user through the interface's control panels, as shown in Figure 2.

By explicitly modeling the interactions between the two spaces, the system introduces a layer of structure that guides the otherwise open-ended generative capabilities of LLMs. The transitions between `C` and `K` are situated within a broader reasoning trajectory, making the design process more interpretable and traceable. The transition engine exposes the four C-K operators as a finite, enumerated set of actions, so that every contribution a user makes resolves to one of them. Each action is bound to a dedicated LLM prompt, keeping model outputs aligned with a specific reasoning task. The engine also tracks dependencies between `C` and `K`, so that when new knowledge is added, it propagates through the tree and the state of every affected concept — approved, rejected, or undecided — is re-evaluated accordingly.

2.4.2 Operationalizing C-K Theory

Responsibility for upholding the C-K framework is distributed across three agents. Structural rules — tree invariants and partition arithmetic — are enforced programmatically by the interface, as these can be checked mechanically and should never be violated. Semantic rules — what counts as a valid concept, and what constitutes a contradiction in `K` — are delegated to the LLM, which can apply judgement where programmatic checks cannot. Final evaluation authority, including the decision to accept or reject a concept, remains with the user.

C-K Theory prescribes both hard structural invariants and softer heuristics for good design practice. We translate the former into **rules**: constraints the system enforces, either programmatically or semantically. We translate the latter into **guiding principles**: heuristics that shape interface and LLM behavior without being mechanically enforced. Together they form the vocabulary used when designing features and when judging whether the system is behaving well.

Rules. Rules are invariants of the C-K representation. They hold at every point in a design session; violating one leaves the representation ill-formed.

1. All knowledge must have a logical status, and those statuses must be mutually consistent.

The user can leverage the LLM to double-check new **K** entries against their own logical status and against the rest of K.

2. The **C** space has a tree structure: the initial concept is the root, and sequential refinement grows the tree.
3. Updates in **K** must propagate through the whole **C** space; concepts whose state changes must be flagged. The LLM re-checks the state of every concept after each **K** update.
4. A concept cannot be edited once it is no longer a leaf in the tree.
5. Knowledge can be added at any time, about any concept, by either the user or the LLM. The LLM can proactively suggest new knowledge to inspire the user.

Guiding Principles. Guiding principles are heuristics about what makes a C-K-guided design session faster, clearer, or more innovative. Some bear on features the interface should offer; others on the LLM's default behavior; others on how the user should engage with the system.

- The more relevant the knowledge a concept is validated against, the faster candidate solutions emerge. The LLM should probe for relevant experiences and populate **K** with information the user may not have.
- The earlier **K** contains the necessary information, the better. **K** management must be future-proof.
- A concept can be refined in several ways, creating multiple exploration paths. The interface must support parallel branches natively.
- The quality of exploration depends on the attributes chosen for refinement. Attribute selection is joint work between the LLM and the user.
- The better organized the **K** space, the easier it is to find relevant information. **K** organization is primarily the LLM's responsibility.
- The clearer the goal encoded in C_0, the clearer the means of attaining it. This is settled during initial concept definition.
- Whether an object is considered completely known depends on the design context. The LLM can support evaluation by highlighting pros and cons; the final decision stays with the human.
- Every LLM suggestion must be logically consistent with existing **K**, and must be substantive — the LLM should not be sycophantic.
- Articulating design concepts precisely is difficult. The LLM can help give form to premature or loosely held concepts.

- Asking and answering questions about **K** is itself a form of K→K expansion. LLMs can generate new knowledge through dialogue with the user.
- Mistakenly evaluating a proposition can have downstream repercussions. Final judgement stays with the human, and LLM reasoning must be transparent.
- User-initiated actions carry more ownership than unprompted LLM suggestions. Users can draft a rough intent and let the LLM refine it before committing.
- Having the LLM raise the right questions is as valuable as having it produce answers.

2.5 Usage Scenario

Two UX designers are tasked with redesigning the onboarding experience for a B2B project management tool with a high early dropout rate. Analytics show that 60% of new users abandon the product within the first session, and exit interviews point to a common complaint: users cannot tell what the tool is for or where to start.

2.5.1 Problem Context

The designers have accumulated a heterogeneous body of user research — interview transcripts, session recordings, heatmaps, and NPS comments — but struggle to synthesize it into a coherent design direction. Past attempts at collaborative brainstorming have produced long lists of disconnected ideas with no clear prioritization or rationale. What is missing is a structured way to move from observed pain points to testable design concepts.

2.5.2 System Application

The designers open the system and define their initial concept C_0: *an onboarding experience that brings a new user to their first meaningful action within five minutes*. This formulation encodes both the goal and a concrete success criterion, grounding all subsequent exploration.

Working individually, each designer partitions C_0 along different attributes. The first designer splits by *guidance modality* — contrasting a fully guided linear walkthrough against a self-directed sandbox mode with contextual hints. The second splits by *personalization depth* — separating role-agnostic onboarding from an adaptive flow that infers the user's job function from their first interactions and adjusts accordingly. The system populates **K** with relevant findings from the research

corpus: studies on cognitive load during first-time use, documented failure modes of tooltip-heavy walkthroughs, and benchmarks from comparable B2B tools.

As each designer evaluates sub-concepts against **K**, the system flags a tension: the adaptive flow requires the user to complete a profiling step that, per the exit interview data already in **K**, is itself a known dropout trigger. The concept is marked undecided pending further partitioning.

The designers then invoke the merge. On the **K** side, the system surfaces a conflict: one designer has logged that progressive disclosure reduces abandonment, while the other has a note asserting that users prefer to see the full feature set upfront. The system presents both propositions with their sources and asks the designers to resolve the contradiction before proceeding. On the **C** side, the two concept trees are integrated: the modality and personalization branches are combined into a unified tree, and the one concept that both designers had marked rejected — a mandatory video introduction — is confirmed as eliminated without reopening the debate.

The merged tree converges on a promising sub-concept: a role-inferred, milestone-driven onboarding flow that surfaces only the three features most relevant to the user's inferred job function, deferring the rest until the first meaningful action is completed. The system evaluates this concept as approved against current **K** and flags it as a candidate solution, while noting two open knowledge gaps — around accessibility requirements and mobile viewport constraints — that the designers may want to resolve before treating the concept as fully specified.

2.6 Key Insights

Several observations emerged from building and using the system. The most unexpected was that structure appears to enable creative work rather than constrain it: unconstrained LLM generation produces shallow diversity — candidates that differ syntactically while resting on similar assumptions — whereas routing generation through explicit **C** $\leftrightarrow$ **K** transitions yields greater conceptual diversity, not less, because each move must be grounded in or must extend the knowledge space.

The role of the LLM also came into question. In this setting the model proved most valuable not as a content generator but as an epistemic agent mediating between uncertainty in **C** and validation in **K** — surfacing gaps, introducing constraints, probing assumptions the designer had not made explicit. Asking it to raise the right questions was often more productive than asking it to propose ideas, shifting the design conversation from ideation-as-output to ideation-as-inquiry.

A less anticipated outcome was that traceability became a deliverable in its own right. Conventional brainstorming leaves the evolution of ideas undocumented, forcing rationale to be recon-

structed after the fact; in the C-K workflow every transition is a record, and the reasoning trace remains available for both reflection and downstream collaboration.

Perhaps the sharpest insight came from the implementation effort itself. Translating C-K into a working system forced a precision the prose alone does not require, and in doing so exposed the points at which the theory goes silent: how to choose partitioning attributes, when to privilege **K** over **C**, how to arbitrate between equally plausible sub-concepts. These are not failures of the theory — they are concrete design questions that the operationalization makes visible for the first time.

2.7 Limitations

Several limitations qualify the contributions described above. The boundary between **C** and **K**, which is sharply drawn in the theoretical formulation, is more difficult to maintain in practice. Generated concepts can smuggle in unverified assumptions, and synthesized knowledge is not always independently verifiable. How **C** and **K** should be defined computationally, and how the system should respond when a contribution falls between them, is itself an open question rather than a settled engineering problem.

Scale is a further constraint. Large concept trees and large knowledge archipelagos need filtering, hiding, and navigation affordances that the current interface does not yet provide. Related questions, such as how to suppress falsified branches without losing their rationale, how to visualize **K** coherently, and how to reuse work from a closed branch in a new context, remain open.

Finally, the system inherits the bias, knowledge, and abilities of its underlying model. The knowledge it surfaces is bounded by the LLM's training data, and that bias can reinforce existing assumptions rather than challenge them, which works against precisely the kind of expansive partitioning that C-K is meant to encourage.

2.8 Future Directions

We see room for more quantitative assessment of partitions along dimensions such as feasibility, cost, and novelty, together with richer explainability affordances as the size of the tree and archipelago grows. Beyond the system itself, the same operationalization strategy could be applied to other design frameworks, which would make it possible to compare how different theories shape LLM-assisted ideation. That comparison should be paired with empirical studies of how

LLM assistance interacts with the divergent and convergent phases of design, and eventually extended beyond text to multimodal concepts that include sketches, diagrams, and physical referents. We plan on conducting a user study to evaluate the system's value in real design workflows. How well C-K Git generalizes across domains, team sizes, and design contexts is something we hope future work will explore.

2.9 Conclusion

C-K Git represents an initial step toward structuring AI-assisted design around a formal theory of reasoning, with the aim of producing design support that is traceable, collaborative, and epistemically accountable. By operationalizing C-K Theory through explicit concept and knowledge spaces, transition operations, and a principled merge workflow, the system offers a way to make design reasoning visible that conventional brainstorming tools do not naturally support. More broadly, C-K Git proposes a design paradigm that uses formal design theory as a scaffold for AI assistance in collaborative, early-stage design

References

A. M. M. Sharif Ullah, Md. M. Rashid, and J. Tamaki, "On some unique features of C-K theory of design," *CIRP Journal of Manufacturing Science and Technology*, vol. 5, no. 1, pp. 55–66, Jan. 2012, doi: 10.1016/j.cirpj.2011.09.001.

A. Hatchuel and B. Weil, "C-K design theory: an advanced formulation," *Research in Engineering Design*, vol. 19, no. 4, pp. 181–192, Jan. 2009, doi: 10.1007/s00163-008-0043-4.

K. Ma, E. R. Brubaker, and K. Goucher-Lambert, "Simulating Design Theory Using LLM Agents: A Case Study of C-K Theory," in *Proceedings of the ASME Design Engineering Technical Conference*, Anaheim, CA, USA, 2025.

J. S. Park, L. Popowski, C. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, "Social Simulacra: Creating Populated Prototypes for Social Computing Systems," in *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*, Bend, OR, USA: ACM, Oct. 2022, pp. 1–18, doi: 10.1145/3526113.3545616.

L. Chen, D. Xia, Z. Jiang, X. Tan, L. Sun, and L. Zhang, "A Conceptual Design Method Based on Concept–Knowledge Theory and Large Language Models," *Journal of Computing and Information Science in Engineering*, vol. 25, no. 2, p. 021001, Feb. 2025, doi: 10.1115/1.4066773.

A. Bordas and A. Pascal, "Bridging Design Science research and formal design theories: leveraging C-K theory for impactful research," in *Proceedings of the 59th Hawaii International Conference on System Sciences*, Honolulu, HI, USA, Jan. 2026, pp. 5254–5263.