

Talking as Theory Reconstruction: Eliciting Architectural Decisions from Developers via Interview and LLM-Based Synthesis

Bernhard Mayr
Storyblok GmbH
bernhard.mayr@storyblok.com

Alexander Feiglstorfer
Storyblok GmbH
af@storyblok.com

Dominik Angerer
Storyblok GmbH
da@storyblok.com

Abstract

In *Programming as Theory Building* (1985), Peter Naur argued that programming is the construction of a theory in the programmer's mind: a theory that cannot be captured in code or documentation, and that dies when its bearers leave the project. The spread of large language models (LLMs), and of agentic development tools that need explicit decision artefacts to operate, makes the claim worth revisiting. This work-in-progress paper reports on a two-stage method for externalising part of the implicit theory behind early-stage software projects. A human interviewer first holds a semi-structured, podcast-style conversation with the developers involved, using open questions to draw out narrative accounts and tacit reasoning. An LLM then distils the transcripts into Architecture Decision Records (ADRs) that record each decision with its rationale, the alternatives, and the trade-offs. To assess fidelity, we cross-check the generated ADRs against the codebase, looking for where stated intent and current implementation match, where they leave gaps, and where they contradict each other. We expect a mixed answer to the question of how much of the theory Naur describes survives reconstruction. Decisions and their trade-offs look partly recoverable; the deeper mental models behind them likely do not. The resulting ADRs may also be useful as context for agentic coding tools, which increasingly need machine-readable rationale to work safely on existing code. We present preliminary findings from two pilot interviews and discuss methodological challenges, among them post-hoc rationalisation, interviewer framing, and LLM hallucination during synthesis.

1. Introduction

Naur (1992) argued that a program is not primarily its source code, but the theory held in the minds of its programmers. The theory connects the program to its problem domain, explains why each part is the way it is, and lets developers respond constructively when requirements change. Code and documentation alone do not recover it. When its bearers leave, it dies.

Forty years later the claim has new practical urgency. Agentic coding tools, AI systems that read, modify, and extend codebases on their own, routinely have to reconstruct rationale that was never written down. In Naur's terms, they arrive like developers who missed all the original meetings: they can read the artefacts, but they do not share the theory.

Architecture Decision Records (ADRs), short structured documents that record a decision together with its context and consequences (Nygard, 2011), have meanwhile emerged as a lightweight way to make rationale explicit. If we can elicit rationale from developers and write it down as ADRs before the knowledge is lost, some part of the implicit theory becomes shared context that future engineers and agentic tools can both read.

This paper asks how much of the implicit theory behind an early-stage software system can be recovered through semi-structured developer interviews and LLM-assisted synthesis into ADRs. We do not expect to refute Naur. We want to locate, in practice, the boundary between what can be reconstructed and what cannot.

2. Background

In Naur's account, the theory a programmer builds has three components. The first relates the program to the real-world domain it models, what we will call the representational layer. The second explains why the program is structured as it is rather than otherwise: the rationale layer. The third gives developers an

anticipatory sense of what would and would not fit the system, a layer that tends to surface only when a new requirement appears (Naur, 1992). Domain documentation and naming conventions recover part of the first layer. The third resists articulation even by the people who hold it. ADRs sit squarely in the second layer, which sets this paper’s scope: we aim to recover the rationale layer ADRs were designed to express, a narrower target than Naur’s theory in full.

ADRs sit within a longer design rationale (DR) tradition of capturing the reasons behind a decision as an artefact in its own right. Early DR notations were formal argumentation structures: IBIS represented a design conversation as a graph of issues, positions, and arguments (Kunz and Rittel, 1970); QOC decomposed a decision into a question, a set of options, and the criteria used to weigh them (MacLean et al., 1991); DRL added explicit goals and claims for richer decision analysis (Lee, 1991). Capturing rationale in such formal graphs proved disruptive alongside design work, a difficulty documented early on as the capture problem (Conklin and Burgess-Yakemovic, 1991). Lightweight, prose-based templates such as ADRs won out over the formal notations for that reason, at the cost of the explicit alternative-versus-criteria structure QOC and DRL required. The interview-based elicitation in this paper is itself a known DR capture strategy, reconstruction: rationale recovered after the fact rather than recorded as decisions are made (Moran and Carroll, 1996). The contribution here is not the capture strategy but pairing it with LLM-based synthesis and a code-level validation pass.

The choice of the word *reconstruction* in the title is intentional. As we argue in Section 5, the interview does not simply retrieve a theory that pre-exists in the developer’s mind. It co-produces an account in dialogue with the interviewer. This limit follows from what an interview, as a dialogic act, produces and cannot be interviewed around.

ADRs are usually written at the moment of the decision. The retrospective case is harder. We are trying to elicit reasoning for decisions made years earlier, much of which was never written down in the first place.

LLM-based coding agents now operate across entire codebases without close human supervision (Hou et al., 2024). They face the same problem Naur described. A codebase contains the outcomes of decisions but not the reasoning behind them. Without the reasoning, an agent cannot tell a deliberate constraint from an accidental one, and may quietly refactor away invariants that exist for reasons not visible in the code. ADRs, even retrospectively reconstructed ones, are written in a form that both developers and coding agents can read. They sit between tacit human knowledge and the explicit context that agentic tools need.

3. Method

The pipeline combines human interviewing with LLM-assisted synthesis, plus a separate validation step against the codebase. The split is deliberate: it keeps each part doing what it is best at, and lets validation catch any drift between what was said and what was actually built. The method has three stages.

Stage 1: Semi-structured interview. A human interviewer holds a semi-structured conversation with one or more developers who were involved in the early stages of the system under study. Questions are open and narrative (“what led to this decision?”) rather than closed (“did you choose X because of Y?”), and the interviewer adapts as the conversation moves (Kvale and Brinkmann, 2009; Patton, 2002). The interviewer is briefed to avoid introducing architectural framings or suggesting alternatives during elicitation. Wherever possible the interview takes place in the participant’s native or strongest language: meaning is constituted in language, and a second language introduces interpretive loss and discourages free expression (Temple and Young, 2004). Interviews are recorded and transcribed. We hypothesise that the interviewer cannot be a complete outsider. Picking out the architecturally significant decisions in a given codebase takes prior exposure to the system, as does knowing which follow-up questions will lead to rationale and not just a restatement of what the code does. In Naur’s terms, the interviewer needs to already share part of the theory; a project-agnostic interview will not get as deep.

Stage 2: LLM-assisted synthesis. The transcript is given to a large language model with instructions to identify the architectural decisions discussed and to format each as an ADR with title, status, context, decision, and consequences. The model is told to flag uncertainty when it is unsure. The specific prompt used in our pilot is shown in Section 4.

Stage 3: Validation. Validation has three steps. First, the interviewer and long-standing team members review the generated ADRs together; the review sharpens shared understanding and improves ADR quality at the same time. Second, where the transcript does not give a date, an LLM reconstructs one from the project’s git commit history, producing a timeline view of how the architecture evolved. Third, the final ADRs are cross-checked against the source code with LLM assistance, to see whether the recorded decisions still hold or whether implementation has since drifted from stated intent.

Separating elicitation from synthesis is intentional. Human interviewers can build rapport, sense when to follow up, and pick up on the unspoken cues a model cannot. The LLM does the tedious work of imposing structure on hours of transcript. Separating the two also keeps the model from steering the interview toward architectural patterns it has seen before.

4. Pilot Study

We ran two pilot interviews. The first lasted around 30 minutes and was conducted by the Storyblok MCP product owner with one of the initial software architects of the Storyblok MCP Server, an open-source integration layer that exposes Storyblok’s content management capabilities to AI coding assistants. The transcript was given to Claude Sonnet 4.6 via Claude Cowork with the following prompt:

I’ve just given you a transcript of an interview we conducted with one of the initial software architects of Storyblok’s MCP Server where we asked him about some architectural decisions that have been made while developing the MCP Server. I would like you to extract these architectural decisions and document them as architecture decision records (in English and markdown). Be as precise as possible and flag uncertainties.

The synthesis produced eleven ADRs. Two are reproduced below as representative examples.

ADR-001: Multi-step workflow (Search → Describe → Execute). Mapping the entire Storyblok Management API to individual MCP tools causes context bloat: every tool description gets injected into the model’s system prompt, burning tokens before any conversation begins. An earlier two-step approach (search → execute) had no confirmation step before execution. The decision was therefore to introduce a three-step workflow: *search* (keyword-based operation discovery, scored by relevance), *describe* (surfaces the full parameter schema for the matched operation before execution), and *execute* (performs the API call). The initial tool footprint stays minimal, with only seven tools exposed at connection time, and operations are discovered on demand. The extra round-trip introduced by the *describe* step is offset by avoiding incorrect executions caused by insufficient knowledge of the parameters.

ADR-003: Role-based endpoint exposure. Different Storyblok users, content editors, developers, administrators, should only see and be able to call the API operations relevant to their role; exposing the full surface to everyone raises the risk of accidental or unauthorized operations. The decision was to accept a *role* query parameter on the MCP server URL that restricts which operations the model can see, enforced consistently across *search*, *describe*, and *execute*, so an operation hidden from search cannot be described or executed either. Embedding the role in the URL itself, rather than in a header or a runtime setting, keeps the restriction transparent to the end user and resistant to accidental override. The trade-off is that adding a role or changing role membership requires updating the OpenAPI overlay and redeploying, rather than a runtime configuration change.

The second interview lasted around 55 minutes and was conducted with one of the co-founders and initial developers of Storyblok, a headless CMS used by large-scale commercial clients. Interviewer and participant spoke their shared native language (Austrian German). The synthesis produced eight ADRs covering the founding and early scaling of the platform. We refer to several of them in the discussion below. One is reproduced here.

ADR-002: Visual Editor V2, full rewrite instead of incremental upgrade. The original Visual Editor (V1) had been written by the two co-founders starting from a pre-stable release of Vue.js. Over several years of growth the surrounding JavaScript ecosystem moved on, and a dedicated frontend team was eventually hired to whom the codebase was, by that point, only fully legible to the two founders who had written it. Rather than upgrading V1 in place and handing it to the new team for maintenance, the decision was to rewrite it from scratch as V2. Two reasons were given together: the Vue.js ecosystem had moved far enough to justify a clean start, and a rewrite would force the new team to build genuine understanding of the architecture, something inheriting a maintained V1 codebase would not have produced on its own. The stated trade-off was the cost of a full rewrite against an incremental upgrade, with the founders' own gloss that it is often easier to rewrite a system than to comprehend an existing one from its artefacts alone.

We are also cross-checking the Storyblok ADRs against the current codebase to see how the method holds up on a long-running production system.

5. Challenges to Validity

Several psychological phenomena limit what an interview-based reconstruction pipeline can produce. We focus on four.

Confabulation and post-hoc rationalisation. Nisbett and Wilson (1977) showed that people have poor introspective access to their own cognitive processes and readily produce plausible causal accounts that do not match the actual causes. When a developer explains an architectural decision years later, the explanation is at least partly reconstructed from current beliefs rather than recalled from the original moment. The MCP Server interview produced a clear example: one decision was described retrospectively as a considered trade-off, but the same interviewee also noted that the relevant aspect of the architecture had not actually been thought about deliberately at the time.

The articulation problem. Polanyi (1966) observed that we know more than we can tell, a claim that lines up with Naur's account of theory. Both hold that expert knowledge is partly tacit and exceeds what any artefact can express. Architectural judgement is partly procedural and resists language. Forcing verbalisation can also distort tacit knowledge: Schooler et al. (1993) documented that verbal descriptions of non-verbal knowledge can degrade subsequent performance and produce accounts that no longer align with the underlying cognition. Whatever the pipeline produces is therefore biased toward the verbalisable subset of the theory.

The observer-construction problem. This is probably the deepest challenge for the paper's framing. The interview does not simply pull a theory out of the developer's mind. It co-produces an account in dialogue with the interviewer. One of the ADRs from the Storyblok interview, documenting a decision about a cache version parameter, only took its shape through the interviewer's clarifying questions about the mechanism. The account did not exist in that form before the interview took place. The same partial sharing of theory that we treated as a precondition in Stage 1 reappears here as a source of bias: the interviewer in both pilots already knew enough about the system to ask useful follow-up questions, but that prior knowledge also shapes what the account ends up looking like. This is why *reconstruction* is more honest than *extraction*: the artefacts the method produces are shaped by the conversation, not by verbatim transcripts of dormant mental models. Design rationale researchers reached a similar conclusion independently: Carroll and

Rosson (1991) argue for scenario-based accounts of design reasoning over formal argumentation graphs, on the grounds that rationale elicited after the fact is argued into shape in the telling rather than retrieved intact.

Illusory theory building. The first three challenges concern what the pipeline cannot recover. The fourth concerns what it may seem to have recovered when it actually has not. Clayton (2026) puts it bluntly: “The issue is that it’s easy to convince yourself that you’ve built theory when you haven’t.” The same risk applies to retrospective reconstruction. A plausible-looking ADR can create the impression of recovered rationale when the developer’s account was shallow to begin with. The illusion of explanatory depth (Rozenblit and Keil, 2002) hits hard in agent-assisted development: a developer who accepts coding agent output without close scrutiny can end up feeling they understand the code better than they do, and may then explain decisions with that inflated sense of certainty in the interview. The Dunning–Kruger effect (Kruger and Dunning, 1999) adds a second layer: limited awareness of one’s own knowledge gaps inflates confidence in the resulting account. A third mechanism is cognitive offloading: handing synthesis to an LLM can itself produce an illusion of competence, since users systematically overestimate their understanding of AI-generated outputs (Jose et al., 2025). We therefore treat the output of the pipeline as a provisional artefact that needs expert review, not a faithful reconstruction.

These challenges do not invalidate the approach. They define what the method actually produces: partial and dialogically shaped approximations of the implicit theory, biased toward what can be put into words. Whether such approximations are useful, whether for engineers onboarding to a codebase, for agentic tools that need rationale, or for preserving institutional memory, is an empirical question. We turn to it next.

6. Discussion

The pilot interviews suggest that decisions and their trade-offs are *partly* reconstructible through this pipeline. Several ADRs generated from the MCP Server interview contain specific and cross-checkable claims: runtime observability data, alternative designs considered and rejected, downstream constraints that cascade from a single decision. These go beyond what reading the code alone surfaces.

Practitioners have made a similar point about agentic tooling: Ekrem (2025) argues that as reflexive AI use lets developers accept generated code with little cognitive engagement, ADRs become one of the few documentation forms that LLM-based coding agents can actually use to recover design intent.

Deeper aspects of the theory Naur describes, the anticipatory judgements that constrain the design space before any decision is framed and the sense of what would and would not “fit” the system as a whole, did not surface clearly in the pilots. We do not expect them to. Where the boundary sits between the reconstructible and the irretrievable is something we are still working out.

The pilots also surfaced a finding that matters in practice: knowledge fragmentation accelerates quickly as teams grow. The Storyblok interview revealed that several decisions made years earlier were already unknown to current team members, and that later, incompatible revisions had been made in internal tooling without any cross-reference to the original rationale. Even a few years of team turnover seem to produce significant gaps. The method is most useful while the original decision-makers are still around.

One observation from the Storyblok interview is a direct instance of Naur’s central claim. The founders described how, once the first dedicated frontend team was hired, continuing to develop V1 of the visual editor turned out to be impractical. It was easier to let the new team build V2 on current technologies than to transfer the theory embedded in V1. This is exactly the scenario Naur predicts: the theory had been lost to the point that rebuilding was preferable to inheriting the existing artefact.

The conference theme of *Juxtapositions* fits the work. The paper sits between human tacit knowledge and developer-readable artefact, and between Naur’s pessimism about documentation and the practical need for AI systems to understand what they are modifying. We are not trying to resolve the tension,

only to map it empirically.

7. Conclusion

We have presented a three-stage method for reconstructing architectural rationale through semi-structured interviews and LLM-assisted synthesis into ADRs, and reported preliminary findings from two pilot studies. The method seems able to produce partial but useful reconstructions of the implicit theory behind early design decisions. Four psychological constraints set principled limits on what it can reach: confabulation, the articulation problem, the observer-construction problem, and the risk of illusory theory building. Our ongoing work is cross-checking the generated ADRs against their respective codebases, and following up on the implications for knowledge preservation practice and for agentic development tools.

References

- Carroll, J. M. and Rosson, M. B. (1991). Deliberated evolution: Stalking the view matcher in design space. *Human-Computer Interaction*, 6(3-4):281–318.
- Clayton, I. (2026). Programming as theory building is true now more than ever. <https://slightknack.dev/daily/2026-02-21/>. Accessed May 2026.
- Conklin, J. and Burgess-Yakemovic, K. C. (1991). A process-oriented approach to design rationale. *Human-Computer Interaction*, 6(3-4):357–391.
- Ekrem, C. (2025). Programming as theory building: Why senior developers are more valuable than ever. <https://cekrem.github.io/posts/programming-as-theory-building-naur/>. Accessed May 2026.
- Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., Luo, X., Lo, D., Grundy, J., and Wang, H. (2024). Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology*, 33(8):1–79.
- Jose, B., Joseph, D., Mohan, V., Alexander, E., Varghese, S. K., and Roy, A. (2025). Outsourcing cognition: The psychological costs of AI-era convenience. *Frontiers in Psychology*, 16:1645237.
- Kruger, J. and Dunning, D. (1999). Unskilled and unaware of it: How difficulties in recognizing one’s own incompetence lead to inflated self-assessments. *Journal of Personality and Social Psychology*, 77(6):1121–1134.
- Kunz, W. and Rittel, H. W. J. (1970). Issues as elements of information systems. Technical Report Working Paper No. 131, Institute of Urban and Regional Development, University of California, Berkeley.
- Kvale, S. and Brinkmann, S. (2009). *InterViews: Learning the Craft of Qualitative Research Interviewing*. Sage Publications, Thousand Oaks, CA, 2nd edition.
- Lee, J. (1991). Extending the Potts and Bruns model for recording design rationale. In *Proceedings of the 13th International Conference on Software Engineering*, pages 114–125.
- MacLean, A., Young, R. M., Bellotti, V. M. E., and Moran, T. P. (1991). Questions, options, and criteria: Elements of design space analysis. *Human-Computer Interaction*, 6(3-4):201–250.
- Moran, T. P. and Carroll, J. M., editors (1996). *Design Rationale: Concepts, Techniques, and Use*. Lawrence Erlbaum Associates, Mahwah, NJ.
- Naur, P. (1992). Programming as theory building. In *Computing: A Human Activity*, pages 37–48. ACM Press / Addison-Wesley. Originally published in *Microprocessing and Microprogramming*, 15(5), 1985, pp. 253–261.

- Nisbett, R. E. and Wilson, T. D. (1977). Telling more than we can know: Verbal reports on mental processes. *Psychological Review*, 84(3):231–259.
- Nygaard, M. T. (2011). Documenting architecture decisions. <https://cognitect.com/blog/2011/11/15/documenting-architecture-decisions>. Accessed May 2026.
- Patton, M. Q. (2002). *Qualitative Research and Evaluation Methods*. Sage Publications, Thousand Oaks, CA, 3rd edition.
- Polanyi, M. (1966). *The Tacit Dimension*. Doubleday, New York.
- Rozenblit, L. and Keil, F. (2002). The misunderstood limits of folk science: An illusion of explanatory depth. *Cognitive Science*, 26(5):521–562.
- Schooler, J. W., Ohlsson, S., and Brooks, K. (1993). Thoughts beyond words: When language overshadows insight. *Journal of Experimental Psychology: General*, 122(2):166–183.
- Temple, B. and Young, A. (2004). Qualitative research and translation dilemmas. *Qualitative Research*, 4(2):161–178.