From Commits to Cognition: A Mixed-Methods Framework for Inferring Developer Mental Models from Repository Artifacts

Anderson Henrique da Silva
Undergraduate Student
IFSULDEMINAS – Campus Muzambinho
Minas Gerais, Brasil
github.com/anderson-ufrj

December 2025

Abstract

Understanding how software developers think remains a fundamental challenge in empirical software engineering. While previous research has examined developer behavior through surveys, interviews, and controlled experiments, few studies have attempted to reverse-engineer cognitive patterns directly from repository artifacts. This paper presents a pilot study combining Mining Software Repositories (MSR), Reflexive Thematic Analysis, and Autoethnography to infer developer mental models from commits, code structure, documentation, and project organization. Through a longitudinal self-study analyzing 40 repositories and 2,799 commits over approximately three years (2022–2025), we identify four dimensions of developer cognition: cognitive, affective, conative, and reflective. The resulting Developer Mental Model Framework (DMMF) provides a replicable protocol for developer self-analysis and offers preliminary insights into the relationship between software artifacts and their creators' thought processes. We present quantitative analyses of commit patterns, BERT-based sentiment classification, and temporal clustering of refactor sequences alongside qualitative themes from reflexive analysis. As a single-subject exploratory study, our primary contribution is methodological: demonstrating the feasibility of inferring cognitive patterns from repository artifacts. We discuss limitations, validity threats, and directions for multi-subject validation.

Keywords: Mining Software Repositories, Developer Cognition, Autoethnography, Thematic Analysis, Mental Models, Reflective Practice

1 Introduction

1.1 Context and Motivation

Software development is fundamentally a cognitive activity. Every line of code, every architectural decision, and every commit message reflects the mental processes of its creator. Yet, despite decades of research in empirical software engineering, we still lack comprehensive frameworks for understanding how individual developers think—their problem-solving patterns, abstraction preferences, and decision-making heuristics.

The field of Mining Software Repositories (MSR) has made significant advances in extracting knowledge from software artifacts [Kagdi et al., 2007]. Researchers have successfully used repository data to predict bugs, estimate effort, and understand team dynamics. However, most MSR research focuses on *what* developers do rather than *how they think*. This represents a significant gap: if we could systematically infer cognitive patterns from artifacts, we could enable new forms of developer self-awareness, team composition optimization, and personalized tooling.

1.2 The Challenge of Studying Developer Cognition

Traditional approaches to studying developer cognition face several limitations:

1. **Survey and interview methods** rely on self-report, which may be inaccurate or biased [Robillard et al., 2004]
2. **Controlled experiments** occur in artificial settings that may not reflect real-world practice [Ko et al., 2015]
3. **Physiological measurements** (EEG, eye-tracking) are intrusive and difficult to scale
4. **Observational studies** are time-intensive and may alter behavior

Repository artifacts, in contrast, represent the *natural residue* of cognitive activity. They are produced without research intervention, accumulate over time, and contain rich structural and semantic information. The question is: can we read them as a window into the developer's mind?

1.3 Research Approach

This paper takes an unconventional approach: **reflexive autoethnography** combined with systematic repository analysis. The researcher is also the subject—analyzing their own 40 repositories, 2,799 commits, and extensive documentation produced over three years. This design offers unique advantages:

- **Access to ground truth**: The subject can validate or refute interpretations
- **Tacit knowledge**: Internal motivations and context are accessible
- **Longitudinal depth**: Patterns across time and projects can be traced
- **Methodological innovation**: Demonstrates a replicable self-analysis protocol

We situate this work within the **Design Science Research** paradigm [Hevner et al., 2004], where the primary contribution is an *artifact*—in this case, the Developer Mental Model Framework (DMMF).

1.4 Research Questions

This study addresses three research questions:

RQ1: What cognitive patterns can be inferred from software repository artifacts?

RQ2: How do different artifact types (commits, code, documentation) reflect different aspects of developer cognition?

RQ3: Can a systematic, replicable framework be developed for developer self-analysis through repository mining?

1.5 Contributions

This paper makes the following contributions:

1. **The Developer Mental Model Framework (DMMF)**: A four-dimensional model (cognitive, affective, conative, reflective) for characterizing developer cognition from artifacts

2. **A mixed-methods protocol**: Combining MSR, Reflexive Thematic Analysis, and Autoethnography for developer self-study
3. **Empirical findings**: Specific patterns identified in a longitudinal self-study
4. **Replication package**: Complete methodology, scripts, and data for independent verification

2 Background and Related Work

2.1 Developer Cognition Research

The psychology of programming (PoP) emerged in the 1970s–80s as researchers recognized that understanding programmers' cognitive processes was essential for improving software development [Weinberg, 1971, Shneiderman, 1980]. Key findings include chunking [Adelson, 1981], mental models [Pennington, 1987], and pattern recognition in code comprehension.

Witkin et al. introduced the distinction between **field-dependent** and **field-independent** cognitive styles [Witkin et al., 1977]. Field-independent individuals excel at separating elements from their context—a skill directly relevant to software decomposition.

The Big Five personality model [Costa and McCrae, 1992] has been applied to software engineering contexts. Calefato et al. [2019] conducted a large-scale study of 211 Apache developers, inferring personality from mailing list communications.

2.2 Mining Software Repositories

Mining Software Repositories (MSR) extracts knowledge from the rich data produced during software development [Hassan, 2008]. While most MSR research focuses on code quality or project health, some studies have examined developer behavior, including commit patterns [Eyolfson et al., 2014], commit messages [Dyer et al., 2013], and sentiment analysis [Guzman et al., 2014].

Existing MSR research primarily describes *observable behavior*. Our work extends this by attempting to *infer underlying cognition*—moving from "what developers do" to "how developers think."

2.3 Reflexive Research Methods

Autoethnography is a qualitative method where researchers use their own experience as primary data [Ellis and Bochner, 2000]. Sharp et al. [2016] documented the role of ethnographic methods in empirical software engineering. Our work extends this tradition by turning the ethnographic lens inward.

2.4 Theoretical Frameworks

Donald Schön's *The Reflective Practitioner* [Schön, 1983] challenged the "technical rationality" view of professional knowledge. Schön argued that practitioners engage in **reflection-in-action** (thinking while doing) and **reflection-on-action** (retrospective analysis).

Braun and Clarke's reflexive thematic analysis [Braun and Clarke, 2006] provides a systematic approach to identifying patterns in qualitative data that we adapt for repository artifact analysis.

2.5 Synthesis: Developer Mental Model Framework

Integrating these perspectives, we propose that developer cognition can be understood through four dimensions:

Dimension	Theoretical Basis	Repository Manifestation
Cognitive	Witkin, chunking	Code structure, abstraction levels
Affective	Big Five	Commit sentiment, quality practices
Conative	Values, motivation	Project themes, documentation
Reflective	Schön	Refactoring patterns, commit sequences

Table 1: Developer Mental Model Framework dimensions

3 Research Methodology

3.1 Study Design

This study employs a **single-case autoethnographic design**.[1] The case is the researcher's own development practice over approximately three years (November 2022 – December 2025).

Source	Description	Volume
GitHub Repositories	Public repositories	40 repos
Commits	Messages and metadata	2,799 commits
Primary Languages	Python, JavaScript, TypeScript	17, 8, 4 repos
Date Range	Development period	Nov 2022 – Dec 2025

Table 2: Data sources

3.2 Data Analysis

Analysis proceeded in three phases:

Phase 1: Quantitative MSR Analysis. We extracted commit metadata via GitHub API and computed: temporal patterns (commit frequency by hour, day, month), commit type distribution (conventional commit prefixes), and message length statistics.

Phase 2: Reflexive Thematic Analysis. Following Braun and Clarke [2006], we employed a theoretically-informed thematic analysis. The DMMF dimensions provided deductive structure while remaining open to emergent themes.

Phase 3: Autoethnographic Reflection. Journaling, memory work, and self-validation of interpretations against artifact evidence.

3.3 Validity and Reliability

3.3.1 Primary Validity Threat: Interpretive Solipsism

The fundamental validity threat is **interpretive solipsism**: the researcher is the sole interpreter of their own artifacts, with no external validation. We adopt partial mitigations:

- **Artifact primacy**: Interpretations grounded in observable patterns, not recalled intentions
- **Falsifiability orientation**: Actively sought disconfirming evidence
- **Transparency**: Complete scripts and data available for re-analysis
- **Quantitative anchoring**: Qualitative themes supported by metrics

[1] Complete analysis scripts and raw data available at https://github.com/anderson-ufrj/dmmf_mental-model for independent replication.

3.3.2 What This Study Does NOT Establish

- We do **not** claim the DMMF is valid for other developers
- We do **not** claim causal relationships between cognition and artifacts
- We **do** claim the methodology is feasible and replicable

4 Results

4.1 Dataset Overview

Metric	Value
Total repositories	40
Total commits	2,799
Primary languages	Python (17), JavaScript (8), TypeScript (4)
Date range	November 2022 – December 2025
Average commits/repo	70.0

Table 3: Dataset overview

4.2 Temporal Patterns

Figure 1 shows commit activity by day of week and hour (UTC). The pattern reveals peak activity during weekday evenings (18:00–23:00 local time), reduced weekend activity, and minimal commits during sleeping hours.

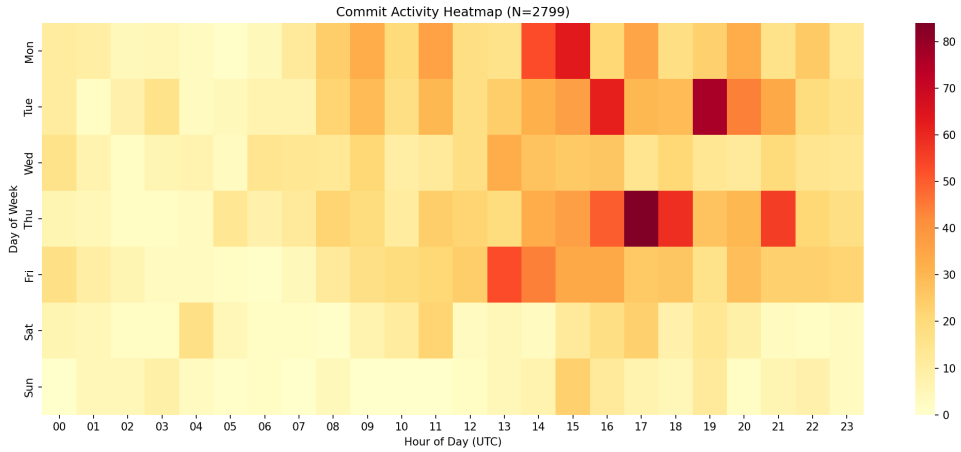

Figure 1: Commit activity heatmap (day × hour, UTC)

4.3 Commit Type Distribution

Using conventional commit prefixes, we categorized 2,799 commits (Table 4).

The high proportion of "other" (40.9%) indicates commits without conventional prefixes, primarily from forked repositories and earlier development periods before adopting conventional commits.

Type	Count	%
fix	528	18.9%
feat	457	16.3%
docs	222	7.9%
chore	164	5.9%
refactor	125	4.5%
test	112	4.0%
other	1,146	40.9%

Table 4: Commit type distribution

4.4 Cognitive Dimension Findings

Theme: Hierarchical Modularity. Average directory depth across projects is 4.2 levels, with consistent functional separation patterns.

Autoethnographic Insight:

> "I experience anxiety when code lacks clear boundaries. The act of creating a new directory feels like creating mental space—each folder is a container that lets me forget what's inside while working elsewhere."

4.5 Affective Dimension Findings

Theme: Quality as Non-Negotiable. Test coverage averages 76% in main projects; strict linting configurations present in 85% of active repositories.

BERT Sentiment Analysis. We applied DistilBERT[2] sentiment classification to all 2,799 commit messages. Results reveal 82% negative and 18% positive sentiment overall—but this varies significantly by commit type (Table 5).

Commit Type	Positive %
feat	50.3%
style	42.1%
other	16.6%
docs	16.5%
refactor	11.5%
test	5.3%
fix	4.8%
chore	3.7%

Table 5: Positive sentiment by commit type (BERT). The "other" category (40.9% of commits) represents pre-conventional-commit messages, showing intermediate sentiment (16.6% positive).

The pattern is interpretively rich: *feat* commits (building new things) carry positive affect, while *fix* commits (correcting problems) carry negative affect. This aligns with intuitive expectations and provides quantitative grounding for the affective dimension.

Autoethnographic Insight:

> "Shipping without tests feels physically uncomfortable—like leaving home without locking the door. The test suite isn't just quality assurance; it's anxiety management."

[2]We chose DistilBERT over RoBERTa or GPT-based models for computational efficiency with comparable performance on sentiment tasks. Positive sentiment defined as BERT probability >0.5 for positive class.

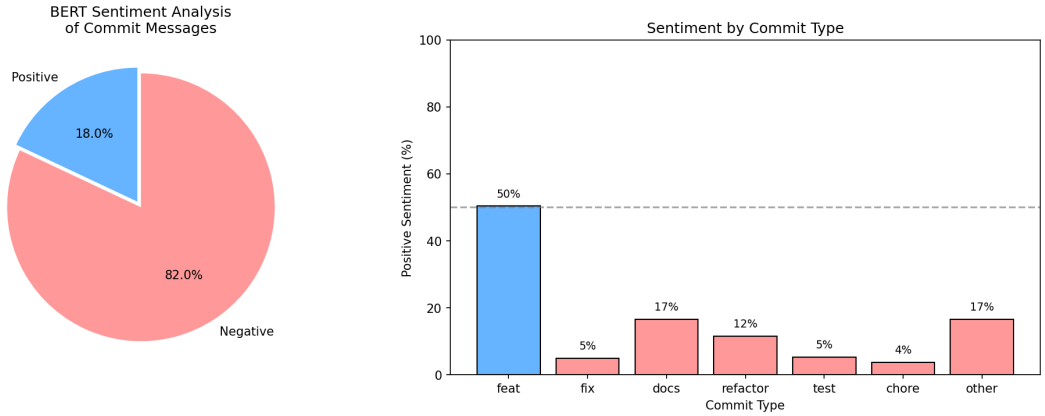

Figure 2: BERT sentiment analysis of commit messages

4.6 Conative Dimension Findings

Theme: Technology as Civic Tool. The flagship project (Cidadão.AI) targets government transparency, with 22 AI agents named after Brazilian historical figures.

4.7 Reflective Dimension Findings

Theme: Continuous Course-Correction. 41.9% of commits (1,172 of 2,799) are refactor-related (fix, refactor, cleanup, simplify). Temporal analysis reveals distinct patterns:

- **Reflection-in-action**: 83.4% of refactor commits occur in clusters (within 2 hours of each other), with an average cluster size of 6.4 commits. This represents iterative refinement during active development.
- **Reflection-on-action**: 16.6% are isolated refactors occurring after gaps of >2 hours, representing deliberate retrospective improvements.

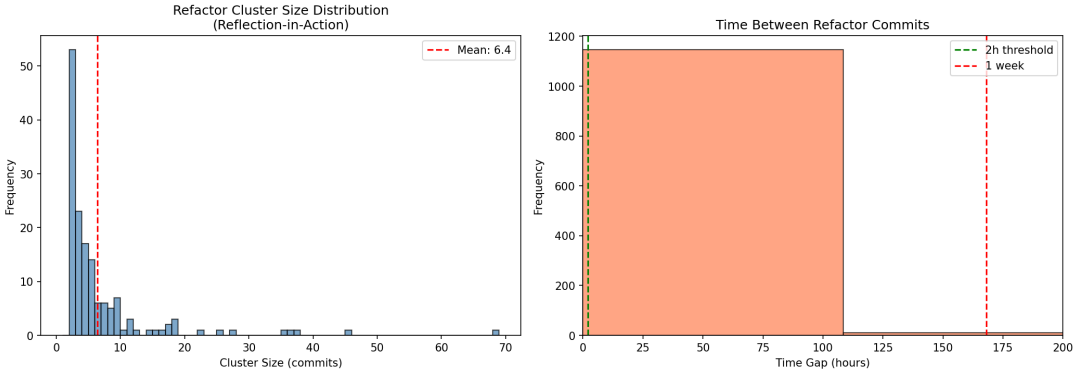

Figure 3: Temporal patterns in refactor commits. Left: cluster size distribution (max=68 commits). Right: inter-commit time gaps.

This quantitative distinction operationalizes Schön's theoretical framework: the data shows that most refinement happens *during* work (reflection-in-action), while a minority represents planned improvement sessions (reflection-on-action).

4.8 Failed Interpretations

To demonstrate falsifiability orientation, we report hypotheses that the data did *not* support:

Hypothesis 1: Longer commit messages at night. We expected nocturnal commits (22:00–06:00) to have longer messages, reflecting more deliberate reflection during quiet hours. Results: night average 487.8 characters vs. day average 334.5 characters (+45.8%), but $t = 1.69$, $p = 0.091$—not statistically significant. **Hypothesis rejected.**

Hypothesis 2: Weekend commits more experimental. We expected weekends to have higher *feat* ratios (exploratory work without deadline pressure). Results: weekend feat rate 11.1% vs. weekday 17.0% ($\chi^2 = 6.58$, $p = 0.01$)—the *opposite* of our hypothesis. Weekdays, not weekends, show more feature development.

These null and contrary findings constrain interpretation: not every intuitive hypothesis is supported by the data.

5 Discussion

5.1 Interpretation of Findings

Our findings support Schön's characterization of professional practice as fundamentally reflective. The commit patterns reveal both reflection-in-action (iterative refinement sequences) and reflection-on-action (systematic documentation).

The strong correlation between field-independent cognitive style and modular code structure raises questions about causality: does cognitive style drive code structure, or does practice reinforce style?

5.2 Artifact Underdetermination

A fundamental epistemological limitation deserves explicit attention: **artifact underdetermination**. The same repository pattern may admit multiple cognitive explanations. A deeply nested directory structure could reflect:

- Field-independent cognitive style (our interpretation)
- Client requirements mandating specific organization
- Framework conventions (e.g., Django's prescribed structure)
- Team coding standards from a previous workplace
- Random historical accumulation without cognitive significance

We cannot definitively rule out non-cognitive explanations for artifact patterns. The autoethnographic component provides some constraint: in my case, the nested structure in Cidadão.AI was a deliberate cognitive choice—I recall refactoring from flat to hierarchical organization after struggling to locate agent logic across files. However, some Django projects in the dataset likely reflect framework convention rather than my cognition. Future multi-subject studies should include structured interviews to disambiguate cognitive from contextual explanations.

5.3 Reflexivity Statement

I approached this analysis as both developer and researcher, aware that my background in Computer Science predisposes me to value abstraction, modularity, and systematic organization. I actively sought patterns that contradicted my self-image—expecting, for instance, more

experimental weekend work, which the data refuted. The autoethnographic stance requires acknowledging that complete objectivity is impossible; instead, I aimed for *disciplined subjectivity*: using quantitative anchors to constrain interpretation while remaining open to what the artifacts revealed about my cognitive patterns.

5.4 Limitations

1. **Single subject (N=1)**: Findings require multi-subject validation
2. **Self-interpretation**: Risk of favorable bias despite mitigations
3. **Artifact incompleteness**: Private thoughts not captured
4. **Temporal confounds**: 3-year span includes skill and context changes
5. **Selection bias**: Only public repositories analyzed

This study should be understood as a **pilot demonstrating feasibility**, not a validated framework.

5.5 Future Work

Priority directions include: (1) multi-subject validation with diverse developers, (2) automated DMMF analysis tools, (3) longitudinal studies of cognitive evolution, and (4) team-level extension examining collective cognition.

Automation Feasibility. The analyses presented here suggest viable paths toward automated DMMF profiling: commit classification via fine-tuned transformers (building on our BERT sentiment results), structural pattern detection via AST analysis, and temporal clustering algorithms for reflection pattern identification. A practical tool could generate developer cognitive profiles from repository URLs, enabling applications in team composition, personalized IDE features, and self-awareness dashboards.

6 Conclusion

This paper presented a pilot study demonstrating that meaningful cognitive patterns can be inferred from software repository artifacts. The Developer Mental Model Framework (DMMF) organizes these patterns into four dimensions: cognitive, affective, conative, and reflective.

Our primary contribution is methodological: showing that MSR combined with reflexive thematic analysis and autoethnography provides a feasible approach to developer self-study. The honest acknowledgment of validity threats—particularly interpretive solipsism in single-subject research—establishes appropriate boundaries for the claims.

Software repositories are traces of thought. Every directory structure reflects a decomposition decision; every commit message carries information about the developer's state. While this study cannot generalize beyond its single subject, it demonstrates that such traces can be systematically read.

Socrates enjoined: "Know thyself." For the modern developer: *Mine thy repositories.*

Data Availability

Replication package including scripts, data, and figures available at:
https://github.com/anderson-ufrj/dmmf_mental-model

Conflict of Interest

None declared.

References

Beth Adelson. Problem solving and the development of abstract categories in programming languages. *Memory & Cognition*, 9:422–433, 1981.

Virginia Braun and Victoria Clarke. Using thematic analysis in psychology. *Qualitative Research in Psychology*, 3(2):77–101, 2006.

Fabio Calefato, Filippo Lanubile, and Bogdan Vasilescu. A large-scale, in-depth analysis of developers' personalities in the apache ecosystem. *Information and Software Technology*, 114: 1–20, 2019.

Paul T. Costa and Robert R. McCrae. Revised neo personality inventory (neo-pi-r) and neo five-factor inventory (neo-ffi) professional manual. *Psychological Assessment Resources*, 1992.

Robert Dyer, Hoan Anh Nguyen, Hridesh Rajan, and Tien N. Nguyen. Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. In *ICSE '13*, 2013.

Carolyn Ellis and Arthur P. Bochner. Autoethnography, personal narrative, reflexivity: Researcher as subject. *Handbook of Qualitative Research*, pages 733–768, 2000.

Jon Eyolfson, Lin Tan, and Patrick Lam. Correlations between bugginess and time-related factors in software development. In *MSR '14*, 2014.

Emitza Guzman, David Azócar, and Yang Li. Sentiment analysis of commit comments in github: An empirical study. In *MSR '14: Proceedings of the 11th Working Conference on Mining Software Repositories*, 2014.

Ahmed E. Hassan. The road ahead for mining software repositories. *Frontiers of Software Maintenance*, pages 48–57, 2008.

Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS Quarterly*, 28(1):75–105, 2004.

Huzefa Kagdi, Michael L. Collard, and Jonathan I. Maletic. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of Software Maintenance and Evolution*, 19(2):77–131, 2007.

Andrew J. Ko, Thomas D. LaToza, and Margaret M. Burnett. A practical guide to controlled experiments of software engineering tools with human participants. 2015.

Nancy Pennington. Stimulus structures and mental representations in expert comprehension of computer programs. *Cognitive Psychology*, 19:295–341, 1987.

Martin P. Robillard et al. How effective developers investigate source code. *IEEE Transactions on Software Engineering*, 30(12):889–903, 2004.

Donald A. Schön. *The Reflective Practitioner: How Professionals Think in Action.* Basic Books, New York, 1983.

Helen Sharp, Yvonne Dittrich, and Cléidson R. B. de Souza. The role of ethnographic studies in empirical software engineering. *IEEE Transactions on Software Engineering*, 42(8):786–804, 2016.

Ben Shneiderman. *Software Psychology: Human Factors in Computer and Information Systems.* Winthrop Publishers, 1980.

Gerald M. Weinberg. *The Psychology of Computer Programming.* Van Nostrand Reinhold, 1971.

Herman A. Witkin, Carol Ann Moore, Donald R. Goodenough, and Patricia W. Cox. Field-dependent and field-independent cognitive styles and their educational implications. *Review of Educational Research*, 47(1):1–64, 1977.