RESEARCH REPORT

Sustained General-Purpose Chat-Native Software Development

History, prevalence, terminology, documentation scarcity, and the artifact accumulation loop

AUTHOR Raynor Eissens

VERSION 1.1
PUBLICATION DATE 28 August 2026
V1.0 DOI 10.5281/zenodo.22140866

RESEARCH ASSISTANCE

Source-driven research and synthesis were conducted with ChatGPT (OpenAI). The report is an independent research record and has not been peer reviewed.

Recommended citation

Eissens, Raynor (2026). Sustained General-Purpose Chat-Native Software Development. Version 1.1. Zenodo. Version DOI assigned on publication.

Abstract

This report investigates sustained general-purpose chat-native software development: a workflow in which a general conversational LLM interface remains the primary control surface for one accumulating executable software artifact across an extended sequence of interactions. The independent search does not support either an extreme rarity claim or a claim of mass but hidden prevalence. Instead, it finds a small but non-empty set of strong historical cases, substantial documentation scarcity, and a structural distinction between ordinary-chat development and modern coding-agent workflows. The report proposes a strict operational definition, a taxonomy separating chat-native work from AI IDEs and coding agents, and the analytical concept of an artifact accumulation loop. It concludes that the practice is probably genuinely narrower and harder to sustain than general AI-assisted coding, while the public record likely exaggerates its rarity because ordinary-chat workflows leave weak provenance and have lacked stable terminology.

Keywords: conversational programming; ChatGPT; chat-native development; AI-assisted coding; coding agents; vibe coding; human-AI interaction; artifact accumulation; software engineering

Version 1.1 correction note

- Version 1.1 corrects the classification of Kloojo. Version 1.0 classified the complete project as Strict, Grade B based on the public methodology statement.
- Post-publication author clarification establishes that Kloojo's core 3D game was developed through sustained ordinary ChatGPT conversation, while the later online multiplayer implementation used OpenAI Codex to work directly with local project files from a prompt/instructions produced in ChatGPT. [86]
- Under this report's taxonomy, current Kloojo is therefore Hybrid overall: a substantial strict chat-native phase followed by a Codex-assisted multiplayer phase. This correction does not change the independent prior-art corpus, definitions, taxonomy, prevalence analysis, or central conclusion.
- Version 1.1 supersedes Version 1.0 (DOI 10.5281/zenodo.22140866). The comparison with Kloojo, GPTGTA, MadeFromChat, and Raynor Eissens still occurs only after the independent corpus and criteria were established.

Full Research Report

Language: English

Core Finding, Definition, and Terminology

A. Executive finding

The independent research does **not** support the strong conclusion that sustained development of serious software through an ordinary ChatGPT-like chat interface almost never occurs. Nor does it support the opposite conclusion that this is a massive but merely poorly visible usage pattern. The public evidence base is simply too thin for either claim.

What the research does show is more specific:

1. The practice demonstrably existed early in the ChatGPT era. In the independent corpus, at least three cases are strong enough to meet a strict definition: Jerry Shi's Sudoku app built iteratively with ChatGPT-4; PublicParkBench's **Circle Clicker**, developed for roughly a year through ordinary ChatGPT and published on Google Play; and **Big 2 Wild**, whose creator reports that roughly 80-90% of the implementation was generated by ChatGPT during a development period of about eight weeks. [1-4]
2. Such cases are **far harder to find in the public historical record than generic "built with AI" projects**. The adversarial/falsification track examined 29 independent candidate cases. Only three could be classified as strict matches without substantially broadening the criteria; three others are especially strong near matches or bounded strict segments. The requested set of 25 well-documented, substantial, sustained, ordinary-chat-first projects could therefore not be validated. That result is evidence of **documentation scarcity**, not evidence that the practice is absent.
3. Conversational iteration in programming is itself certainly not exceptional. A CodeChat study of 82,845 real-world developer-LLM conversations reported that 68% were multi-turn. Another study of shared ChatGPT conversations linked in GitHub pull requests and issues found multi-turn use in roughly one third of the conversations. These datasets therefore show abundant *prompt -> correction -> follow-up prompt* behavior, but they do not answer whether the same executable artifact was developed for weeks or months through ordinary chat. [5, 6]
4. The most important structural distinction is probably not the language model but the **development loop**. Historically, an ordinary chat does not automatically see the current repository, runtime, compiler, browser, test results, and changed files. The human has to synchronize those worlds. With coding agents, the harness performs a substantial part of that work itself: reading the repository, modifying files, calling tools, running tests, and trying again. OpenAI explicitly describes its Codex agent in terms of a harness that orchestrates model, user, and tools in an agent loop; surveys of code agents likewise use autonomy and environment interaction as core distinctions. [7-9]
5. The best explanation for the apparent rarity is therefore **a combination of factors**: ordinary-chat development had real state, context, and synchronization friction; some users subsequently took over coding themselves; others migrated to Cursor, Copilot, Claude Code, Codex, Windsurf, or similar systems; and the remaining chat-native projects were usually not documented methodologically. The emergence of such specialized systems - Windsurf in November 2024, Claude Code in 2025, and Codex as a cloud software-engineering agent in May 2025 - shows that the industry productized precisely the limitations ordinary-chat users had previously handled manually. [10-12]

The most defensible final conclusion is therefore: sustained general-purpose chat-native software development appears to be **a genuinely narrower and harder long-term workflow than general AI-assisted coding**, but probably **substantially less rare than the public historical record suggests**. What is strikingly scarce is not the existence of prompt/test/revision cycles; it is public documentation of a long causal chain in which *ordinary chat* remains the central development medium through most of a growing project.

That conclusion matters all the more because the potential user population is enormous. OpenAI reported more than 900 million weekly ChatGPT users in February and March 2026; a later release note used the looser lower bound "700M+". None of those figures, however, contains the dimensions this study needs: software development, ordinary-chat interface, number of project days, AI share of implementation, manual coding, or agent use. A huge denominator therefore does not make the missing prevalence telemetry any less problematic. [13-15]

B. Definition

The strongest workable definition is:

> **Sustained general-purpose chat-native software development** is a development mode in which a human uses a general conversational LLM interface, over an extended sequence of interactions, as the primary control

> surface for one accumulating software artifact; the human steers primarily through natural language and observed behavior, while the LLM generates or modifies most of the implementation, without a specialized coding agent, AI IDE, or substantial manual programming taking over the dominant development loop.

For a **strict match**, this study uses seven hard gates.

Gate	Requirement
Interface	Ordinary general-purpose conversational LLM chat is the primary AI development interface.
Steering	Human intent is expressed primarily through requirements, feedback, errors, tests, screenshots, or descriptions of behavior.
Implementation	The LLM generates/modifies a substantial majority of the implementation.
Persistence	Not one prompt or a few fixes, but an extended chain in which the same artifact accumulates state.
Artifact	Actual executable software emerges: an app, game, service, website, tool, or system.
No dominant agent/AI IDE	Cursor/Windsurf/Claude Code/Codex/Devin/Replit Agent or an equivalent autonomous environment is not the primary executor.
No dominant manual implementation	The human may copy, deploy, test, manage files, and make small fixes, but does not write so much implementation that the LLM becomes merely a conventional assistant.

Complexity, duration in calendar days, deployment, and multimodality are important **gradations**, not absolute gates. An intensive twelve-hour session with dozens of test/rewrite cycles can therefore, in principle, be sustained; a six-month project in which ChatGPT is only occasionally asked for help is not.

This choice aligns with an existing research distinction between one-shot code generation and conversational interaction. Ross et al. explicitly emphasized in 2023 the importance of preceding conversational context and the "artifact under development," and demonstrated extended multi-turn software conversations. Their Programmer's Assistant, however, was a research prototype, not evidence of ordinary ChatGPT serving as a sustained production environment. [16, 17]

C. Recommended terminology

No established term found in this research simultaneously encodes the four defining axes: **ordinary general-purpose chat + sustained duration + artifact accumulation + non-agentic human control**.

Term	Relationship to target category	Problem
Conversational programming	Best academic umbrella	Also includes specialized prototypes, professional programmers, and IDE chat. Ross et al. use conversational interaction much more broadly. [16]
Conversational software development	Good broad practitioner term	Appeared early, but has no stable operational boundary around interface or duration. [18, 19]
Chat-oriented programming / CHOP	Important adjacent term	In practitioner use, CHOP also includes specialized coding chat and developer tooling, so it is broader than ordinary-chat-only. [20, 21]
Natural-language programming	Historical/conceptual umbrella	Includes program synthesis, NL specifications, and other methods without conversational project persistence. [7]
Prompt-driven programming	Overlap	Can mean one prompt, builders, or agents.
AI pair programming	Adjacent	Often implies that the human programmer continues to code/review; GitHub already used "AI pair programmer" for Copilot in 2021. [22]
Vibe coding	Strong overlap in natural-language steering	Interface-agnostic, often short/prototypical or agent/IDE-based, and originally associated with minimal code review. [23-25]
Agentic programming	Structurally different loop	An agent plans, acts, tests, and corrects

Term	Relationship to target category	Problem
		with tools under limited continuous human intervention. [7, 8]
Sustained general-purpose chat-native development	**Recommended strict term**	Descriptive, neutral, and makes no historical "first" claim.

For that reason, the formulation used by the research brief itself is surprisingly strong. I recommend retaining **sustained general-purpose chat-native software development** for the strict class, with **sustained chat-native development** as an informal shortening. "Conversational programming" can serve as the broader academic family.

"Chat-native" is essential here: it indicates that conversation is not merely a feature alongside the IDE, but the place where requirements, corrections, and subsequent implementation steps are primarily formulated.

Taxonomy and Research Discipline

D. Taxonomy

The best taxonomy does not follow the model being used. Instead, it asks **who controls the development loop, where project state is held, and who performs actions in the environment**.

Mode	Dominant loop	Who writes code?	Who sees/runtime-tests the environment?	Typical example	Relationship to strict
One-shot AI code generation	Prompt -> code	LLM	Human	"Make a Snake game"	Excluded
Short conversational prototyping	Prompt -> test -> a few fixes	Mostly LLM	Human	app in 20 min./1 hour	Near, but not sustained
Sustained conversational development	Many repeated prompt/test/revision cycles	Mostly LLM	Mostly human	Circle Clicker	Potentially strict
AI-assisted conventional programming	Human writes and AI helps	Human + LLM	Human/IDE	debugging, snippets, review	Excluded
AI pair programming	Joint implementation	Mixed	Human/IDE	Copilot pair programming	Usually excluded
AI IDE development	Conversation inside a codebase-aware IDE	Much AI	IDE + human	Cursor/Windsurf	Excluded
Vibe coding	Intent -> AI output -> try -> adjust	Often mostly AI	Varies	Karpathy-style	Overlapping, not identical
Coding-agent development	Task -> agent reads/writes/tests	Agent	Agent tools	Claude Code/Codex	Excluded
Autonomous agent development	High-level goal -> long independent execution	Agent	Agent	Devin	Excluded
Multi-agent development	Human delegates to multiple agents	Agents	Agents	agent swarms	Excluded
No-code/AI app builder	Prompt -> platform generates/hosts app	Builder	Builder/platform	Replit Agent/Lovable-like	Excluded
Hybrid workflow	Switches among chat, agent, IDE, and manual code	Mixed	Mixed	ChatGPT -> Cursor	Keep separate

This distinction prevents an important problem that became increasingly pronounced from 2025 onward: two people can both say "I program by talking to AI," while the first manually moves files back and forth between ChatGPT and a runtime and the second instructs an agent to inspect a repository autonomously, modify files, run tests, and prepare a pull request. The linguistic surface looks similar; the HCI architecture is radically different. OpenAI, for example, explicitly describes Codex as a cloud software-engineering agent operating on a repository in a sandbox; Claude Code was introduced as a terminal tool to which "substantial engineering tasks" can be delegated; Devin was given a shell, editor, and browser in its own compute environment. [10, 12, 26]

Karpathy's original description of *vibe coding* from 2 February 2025 captures a different axis: it emphasizes barely reading diffs, returning errors to the AI, and proceeding largely on the basis of results and feel. Later academic and industry definitions became broader still, ranging from trial-and-error without full code comprehension to almost any natural-language-first app development. A strict chat-native project can therefore be vibe coded, but does not have to be; conversely, a vibe-coded Cursor, Claude Code, or Replit project is not a strict general-purpose-chat case under this study. [23-25, 27]

Research protocol. The comparison with Raynor Eissens, Kloojo, GPTGTA, and MadeFromChat was examined only after the independent corpus, criteria, and preliminary classifications had been established. The adversarial track explicitly aimed to break the rarity hypothesis. Rather than searching only for terminology, the research followed claims such as "100% coded with ChatGPT," "no coding experience," "all code by ChatGPT," "built an app using ChatGPT," month- or year-long projects, deployment claims, and migrations to other tools.

Evidence grading

Grade	In this study
A	Direct build logs, conversation screenshots/transcripts, repository history, videos, or other contemporaneous process evidence from which the iterative loop can be reconstructed well.
B	Detailed first-person retrospective description plus an identifiable artifact, but without a complete audit trail.
C	Credible secondary or partial description, insufficient to establish interface/code share/duration completely.
D	"Made with AI/ChatGPT" claim, marketing text, or README without enough process information.

Importantly, **primary source** and **Grade A** are not the same thing. A creator's GitHub README can be a primary source and still be Grade D when it says only "built entirely with ChatGPT." Conversely, a long first-person development account on Reddit can be methodologically much more valuable when it precisely describes how much code the human wrote, how context was restored, and how often the artifact was tested.

Historical Development and Conceptual Prehistory

E. Historical timeline

Period	Development	Significance for chat-native development
2022	ChatGPT is publicly launched on 30 November. Soon afterward, project claims already appear, such as a VS Code extension "built using only ChatGPT." The phrase "conversational software development" also appears almost immediately in practitioner discussion. [18, 28-30]	For the first time, a broadly accessible general chat interface becomes practically useful as a code generator. Project management, repository state, and execution remain entirely outside the chatbot.
2023	Ross et al. publish *The Programmer's Assistant*, explicitly addressing extended multi-turn conversation grounded in code. GPT-4 appears in March. ChatGPT later gains Code Interpreter/file interaction and then image input. Early first-person projects such as Write Like Native and the Sudoku build show real test -> feedback -> rewrite loops. [1, 16, 31-34]	Conversation becomes more than Q&A. The human nevertheless remains the manual bridge among chat, IDE, compiler, files, and browser.
2024	Circle Clicker is published on Google Play	2024 is an inflection point: just as ordinary-

Period	Development	Significance for chat-native development
	after roughly a year of ordinary-ChatGPT development; Big 2 Wild describes eight weeks of intensive ChatGPT -> IDE -> test -> ChatGPT work. Another creator documents roughly 200 hours and explicit context-management techniques. OpenAI introduces Canvas as an interface for writing and coding projects "beyond simple chat," and later Projects for shared chats/files/context. Windsurf appears as an AI-native IDE and Devin marks the agentic direction. [2-4, 11, 35-38]	chat builders publicly describe state problems, products emerge that bring project state directly into the AI environment.
2025	Karpathy coins *vibe coding*. Claude Code appears as an agentic terminal tool; OpenAI launches Codex as a cloud software-engineering agent. ChatGPT memory expands to use more previous chat history. CodeChat finds substantial multi-turn coding dialogue. GitHub further develops multi-file conversational editing. [5, 10, 12, 25, 39, 40]	Natural-language software creation gains enormous cultural visibility, but the dominant tooling shifts from detached chat to repository-aware environments. Ordinary-chat development therefore becomes harder to recover as a distinct historical category.
2026	OpenAI reports more than 900 million weekly active ChatGPT users in early-2026 publications. *Programming by Chat* analyzes conversational programming at large scale, but explicitly in an **IDE-native** setting: 74,998 developer messages, 11,579 chats, 1,300 repositories, and 899 developers from Cursor/GitHub Copilot contexts. OpenAI integrates Codex ever more deeply into ChatGPT as an agentic execution layer. [14, 15, 41, 42]	"Chat" and "agent" are now combined even at the UI level inside the same brand. It therefore becomes even more important to treat the development loop, rather than the product logo, as the unit of analysis.

The product history supports H9 - "general chat was never culturally framed as a development environment" - fairly strongly. ChatGPT began as a conversational product; Code Interpreter/files followed in 2023, Canvas only in October 2024, and Projects in December 2024. OpenAI now explicitly describes Projects as suitable for multi-session and long-running work. These are precisely the affordances a persistent development environment needs, but they were only gradually added to the original chat. [30, 31, 36, 37, 43]

Multimodality changed the loop as well. From 2023 onward, ChatGPT could interpret images, making screenshots usable as a feedback channel alongside textual error reports. This reduces the translation cost between "what the human sees in the running app" and "what the model needs to know." [32]

The evolution of context must be described cautiously. API model context and the practically usable state of the ChatGPT interface are not the same thing. There is, however, direct user evidence that the limitation was historically felt: the creator of Circle Clicker wrote that things became difficult once files were linked together and attributed the later feasibility of the project partly to increased context capacity. Another creator advised in 2024 keeping files smaller than a single ChatGPT response, re-uploading changed files, making backups, and returning to an earlier working state after "bad iterations." [2, 38]

Conceptually, parts of this workflow existed before LLM chat: program synthesis turns intent into programs, mixed-initiative co-creation studies shared human-machine initiative, and end-user programming studies programming by non-specialists. What converges from 2022 onward is a general-purpose language interface that can simultaneously discuss requirements, write code, interpret bugs, and use preceding dialogue as local project context. The literature on code agents explicitly places classical program synthesis as a precursor to modern LLM code generation. [7, 44]

Independent Prior-Art Corpus and Falsification

F. Prior-art corpus

The table below contains the 29 cases included before the comparison with Kloojo/GPTGTA/MadeFromChat. "Strict" means the hard gates are sufficiently supported; "near" means that one crucial boundary in particular remains uncertain; "hybrid" means the workflow demonstrably transitioned to other AI development tooling.

Case	Start	Artifact	Interface / implementation	Duration	Deployment	Evidence	Classification
barnesoir ChatGPT VSCode extension	Dec. 2022	VS Code extension	"only ChatGPT"; process otherwise unknown	unknown	repo/marketplace	D [28, 29]	Excluded
Reddit countdown web app	Jan. 2023	Replit countdown	ordinary ChatGPT, non-programmer	short	public demo	B/C [45]	Excluded: short
Aris Catsambas game	2023	simple letter/game prototype	ChatGPT	short/unknown	demo	B/C [46]	Excluded
Jerry Shi Sudoku	GPT-4 Plus era; page not explicitly dated	iOS Sudoku with generator, validator, UI, input/hints	ordinary ChatGPT-4; AI writes code/tests, human runs and reports; later small manual fixes	approx. 12 hours, multiple sprints	app/shipping trajectory	**A** [1]	**Strict**
Sasha Petrov, Write Like Native	May 2023	Vue/Flask web app	ChatGPT-4 for entire cycle; full conversation was shared	8-10 hours	public web app	A/B [34]	Near: short/simple
Daniel Manzke script	2023	technical script	ChatGPT, several correction rounds, partly modified manually	short	script	B [47]	Excluded
hupka scan-pass	2023	C++ training app	"built entirely with ChatGPT"; process unknown	unknown	GitHub	D [48]	Excluded
Reddit: macOS app, Slackbot, PDF tools, scrapers	2023	multiple tools	non-programmer says they were built with ChatGPT	unknown	unknown	D [49]	Excluded: insufficient reconstruction
Reddit: two apps/cloud architecture	2023	two apps	GPT-4 guides architecture; creator can program	months	unknown	C/D [50]	Excluded: possibly conventional AI-assisted development
Slack PII Detection Bot	2023/24	Node Slack bot	README claims code/config/README by ChatGPT	unknown	GitHub/runnable	D [51]	Excluded: no sustained evidence
PublicParkBench, Circle Clicker	approx. 2023-Apr. 2024	Flutter/Firebase Android game	no coding experience; claims 100% code by ChatGPT; small chunks, one task per chat	**approx. 1 year**	**Google Play**	**B** [2, 3]	**Strict**
Big 2 Wild	2024	React Native mobile card game	ChatGPT -> VS Code -> test -> back; approx. 80-90% AI code, 10-20% manual	**approx. 8 weeks**	**App Store + Play Store**	**B** [4]	**Strict**
coaststl platform	2024	platform for agent-autonomy testing	ordinary ChatGPT primary; file uploads, modularization, backups; Copilot/Groq also used for debugging	approx. 200 hours; 6 months incl. detours	prototype	B [38]	Hybrid / near
One-hour iOS Tetris	2024	Tetris-like app	ChatGPT, framed as no prior coding experience	approx. 1 hour	working demo	B/C [52]	Excluded: short
PropDecks	unknown	fantasy-football app	public claim of entirely ChatGPT prompting, but	unknown	website + Play Store	D [53-55]	Excluded

Case	Start	Artifact	Interface / implementation	Duration	Deployment	Evidence	Classification
			method cannot be reconstructed				
Ann Jackson calendar assistant	2025	calendar assistant	Custom GPT + Actions	unknown	operational	B/C [56]	Hybrid/builder
Scott Knowles TikTok scraper	2025	OCR/transcript/scraper/ Supabase app	ChatGPT code, but later VS Code/Codex tooling	unknown	working app	B/C [57, 58]	Hybrid
Markolé SaaS	2025	full-stack SaaS	Gemini + ChatGPT/mixed workflow	approx. 9 weeks	public product	B [59]	Hybrid
No-code app, ChatGPT + Cursor	2025	app	ordinary chat and Cursor	approx. 5 months	unknown	B/C [60]	Hybrid
Workout app, ChatGPT/Claude -> Cursor	2025	workout app	multiple models, then AI IDE	approx. 5 months	working	B [61]	Hybrid
The Focus Project	2025	desktop app	AI helped heavily but creator ultimately had to learn programming	sustained	desktop demo	B [62]	AI-assisted conventional/hybrid
GPT-5 Canvas travel app	2025	two travel-app versions	ChatGPT Canvas, almost no code	approx. 25 min.	functional	B/C [63]	Excluded: short
Nick Radcliffe, month-long CHOP	2025	large test-driven project, approx. 1,500 tests	>99% AI-code claim	approx. 1 month	working	B [64, 65]	**Excluded: Claude Code agent**
Karpathy vibe-coding workflow	Feb. 2025	experimental software	natural language, errors returned, little diff reading; tool-oriented workflow	short iterative sessions	experiments	B/C [25, 27, 66]	Excluded: vibe/AI-tool class
ChatGPT -> Codex Android game	2026	Android game	ChatGPT as tutor/project manager; then Codex in VS Code	sustained	Google Play	B [67]	Hybrid
Christian Ortega market tracker	2026	searchable farmers-market tracker	had never written code; built with ChatGPT	one afternoon	business website	B [68]	Excluded: short
Dan McInerney MMA prediction system	chat phase approx. 2023-26	database, complex SQL, ML features, model/site	says he wrote no line of code himself for approx. three years; copy/paste from ChatGPT 3.5 -> 4/4o; extensive debugging	**approx. 3 years for claimed phase**	public/open-source components	**B** [69]	**Near / strict segment**
Swift-beginner calculator/memo progression	2025	small Swift apps	ChatGPT writes code	approx. one week	local	B/C [70]	Excluded: separate learning exercises
Financial analyst: five small AI apps + larger app	2026	work tools + unfinished app	AI tool/interface cannot be reliably established	approx. 5 months for larger app	small internal tools; main app unfinished	C/D [71]	Excluded: interface/unfinished

G. Strict-match corpus

The independent strict-match corpus is therefore remarkably small, but not empty.

Strict case	Why it crosses the boundary	Caveat
Jerry Shi Sudoku	Ordinary ChatGPT-4; many distinct conversational test/fix cycles; AI generates core classes, tests, and UI; human runs Xcode and gives feedback; artifact grows sprint by sprint. [1]	Creator is an experienced developer in other languages and later repairs small errors himself; total calendar duration is short.
Circle Clicker	No coding experience, explicit 100%-ChatGPT-code claim, sustained for about one year, linked-file/context problems, database/login/ads/gameplay, and public Play Store release. [2, 3]	App remains technically relatively simple; no complete conversation export. Grade B.
Big 2 Wild	Eight weeks; ordinary ChatGPT + ordinary editor; 80-90% generated code; iterative rewrite/fix loop; production mobile release on two stores. [4]	10-20% manual code and external product design; therefore less "pure" than Circle Clicker, but AI clearly remains the primary implementer.

The strongest **near match** is Dan McInerney's MMA/UFC system. It is substantially more complex and longer-running: complex SQL, feature engineering, models, and a public site; he describes years of copy/paste programming from ChatGPT and says he wrote no code himself during that phase. But the overall project history began before the ChatGPT phase. It therefore cannot responsibly be claimed that the entire artifact was strict chat-native from scratch; a substantial multi-year development phase does, however, appear to meet almost all strict criteria. [69]

Write Like Native sits at the opposite edge. The workflow is exceptionally well documented and the creator even made the entire ChatGPT conversation available, but he described the project itself as relatively simple and completed it in eight to ten hours. This is the strongest argument for **not defining duration purely as a number of hours**: methodologically it is nearly the same pattern, but historically it is more prudent to call it "short-but-deep conversational prototyping." [34]

H. Near matches and exclusions

The main exclusion patterns are structural, not arbitrary:

Exclusion class	Examples	Why not strict
Too short	countdown, Tetris in an hour, GPT-5 travel Canvas, Christian Ortega tracker	Shows code generation and several refinements, but not sustained accumulation. [45, 52, 63, 68]
"Built with ChatGPT" without a process trail	VSCode extension, scan-pass, Slack PII bot, PropDecks	Code authorship is insufficient to reconstruct duration, interface, agent use, and human coding. [28, 48, 51, 55]
Manual programming becomes important	Focus Project; parts of Jerry Shi/coaststl	The LLM then becomes more of an assistant/pair than the primary implementer. [1, 38, 62]
Migration to AI IDE/agent	ChatGPT -> Cursor cases; ChatGPT -> Codex Android game	The later causal chain is carried by a different development system. [60, 61, 67]
Agent from the start	Nick Radcliffe/Claude Code	Highly relevant as natural-language development, but exactly the kind of tooling excluded by the strict definition. [64, 65]
Builder/custom platform	Custom GPT + Actions; Replit/Lovable-like workflows	The platform does more than conversational code generation and manages environment/state. [56, 72]
Multi-model/mixed stack	Markolé, ChatGPT+Claude+Cursor	It is no longer possible to isolate what general-purpose chat did as the development environment. [59, 61]

This is also why a literature search for "vibe coding" after 2025 can paradoxically become worse for the original research question: results are flooded by Replit, Cursor, Claude Code, Copilot, and agentic workflows that look similar at the level of natural language but are fundamentally different at the level of the development loop. GitHub, for example, described vibe coding in 2025 as a route from idea directly to a runnable proof of concept, while its own tooling naturally includes Copilot/context-aware code editing. [24, 39]

O. Strongest falsification result

The strongest falsification of the claim that "this almost never happened before the modern coding-agent era" is **Circle Clicker**. The creator started with virtually no coding experience, sustained the same method for roughly a year, used small ChatGPT code chunks, divided tasks across chats because of context loss, had ChatGPT guide him through Flutter, Dart, and Firebase, added persistent scores/login/monetization, and actually released the result on Google Play. This is exactly the kind of behavior that, under a strong rarity hypothesis, should have collapsed after a few days or weeks. [2, 3]

The second falsification is collective: CodeChat shows that **iterative conversational correction** can itself be normal. Rarity therefore cannot simply be explained by saying that "programming users usually ask one question and stop." Sixty-eight percent of the examined developer conversations were multi-turn. On the other hand, multi-turn says nothing by itself about a project spanning hundreds or thousands of iterations. [5]

The adversarial track has therefore **falsified** a stronger version of the rarity hypothesis - "ordinary chat cannot function for an extended period as a primary programming environment" is untenable - while the weaker question, "how often do people keep doing this for months rather than migrate?" remains open.

Prevalence, Visibility, and Why It Appears Rare

I. Prevalence

On the basis of public material, there is **no defensible percentage** for strict sustained general-purpose chat-native development.

At least four measurement variables are missing from existing telemetry and datasets: whether the interface was ordinary chat or a coding harness; whether conversations belonged to the same artifact/project; how much manual code the human wrote; and whether an agent/AI IDE became the primary developer at any point.

DevGPT-like research is extremely useful but methodologically selective: one study examined 580 shared ChatGPT conversations explicitly linked in GitHub pull requests and issues. DevChat later collected 2,547 unique shared ChatGPT links on GitHub from May 2023 through June 2024. These are valuable traces of real developers, but by construction they include only conversations that someone publicly linked to GitHub. A months-long private ChatGPT project with no shared chat links disappears entirely from such a corpus. [6, 73, 74]

WildChat and CodeChat address a different part of the problem. WildChat collected a very large corpus of real ChatGPT interactions; CodeChat extracted 82,845 developer-LLM conversations and hundreds of thousands of code snippets from it. But here too, the unit is generally the **conversation/session**, not a project identity that resumes the same executable artifact months later. [5, 75, 76]

The newer *Programming by Chat* study moves much closer to genuine project context - 1,300 repositories and 899 developers - but specifically studies Cursor and GitHub Copilot, making it IDE-native conversational programming. This is valuable for the broader phenomenon and simultaneously illustrates the historiographic gap: once research can observe conversational programming at project level, the infrastructure has often already become a specialized IDE. [41]

J. Why it appears rare

Hypothesis	Assessment	Evidence for	Evidence against / limitation
H1 - Not rare, only undocumented	**Plausible, medium-high**	Large datasets depend on shared links and selected public traces; many discovered cases exist only as isolated Reddit posts/READMEs. [6, 74]	There are strikingly few detailed month-/year-long B/A-grade logs despite intensive searching.
H2 - Short-session usage dominates	**Mixed**	GitHub-linked conversations in one study were usually not multi-turn. [6]	CodeChat found 68% multi-turn. Multi-turn is therefore not rare; project-long persistence remains unknown. [5]
H3 - Context/state friction creates a ceiling	**Strongly supported**	Circle Clicker mentions linked files/context; coaststl recommends small modules, file re-uploads, backups, and rollback. Agent research notes that native LLMs lack autonomous cross-file/environment loops. [2, 7, 38]	New memory, Projects, files, and larger context reduce this problem. [40, 43]
H4 - Users manually take	**Moderately supported**	Jerry Shi began repairing small	Circle Clicker shows that

Hypothesis	Assessment	Evidence for	Evidence against / limitation
over		errors himself; the Focus Project says AI ultimately forced the creator to learn programming; coaststl sometimes recommends fixing things manually. [1, 38, 62]	exceptionally persistent users can continue without such a takeover. [2]
H5 - Users migrate to specialized tools	**Strongly supported**	Several first-person cases move from ChatGPT to Cursor or Codex. Specialized tooling expands rapidly at the same time. [10, 60, 61, 67]	Public migration cases do not provide a reliable migration rate.
H6 - Artifact accumulation is cognitively demanding	**Strong qualitative evidence**	A year for a simple game; 200 hours plus manual state management; explicit requirement decomposition and version snapshots. [2, 38]	No population study measures this directly.
H7 - Low cultural visibility	**Medium-high**	Research/marketing is organized around tools, models, and agents; the ordinary-chat workflow has little distinct branding or repository trace.	Cultural "visibility" is difficult to operationalize quantitatively.
H8 - Terminology erases the distinction	**Strongly supported**	Conversational programming, CHOP, and vibe coding all have broader meanings; vibe coding is even used for AI IDEs/agents. [16, 21, 24, 25]	A broad term can also increase the chance that cases become discoverable under one umbrella.
H9 - General chat was not framed as a development environment	**Strong historical support**	OpenAI introduced Canvas in 2024 explicitly for work that went "beyond simple chat"; Projects arrived only at the end of 2024 as shared context for long-running work. [36, 37, 43]	Users could of course repurpose the interface as a development environment even before that product framing.
H10 - Survivor/search bias	**Highly plausible, not quantifiable**	People call projects "made with ChatGPT," "100% coded by ChatGPT," "AI built," etc.; such labels omit the required methodological metadata.	An undiscoverable population cannot, by definition, be directly measured through web search.

The cases also suggest two additional hypotheses.

H11 - the manual orchestration tax. With ordinary chat, the human is not only product owner but also a primitive software-engineering harness: copying code, choosing the correct file, reproducing an error, collecting console output, taking a screenshot, returning the current version, explaining what broke, replacing changed code, testing again, and rolling back when necessary. Jerry Shi described the problem sharply: ChatGPT could not "see" what the IDE saw or run the program itself, so he had to translate runtime feedback into conversation. [1]

Coding agents are, to a large extent, an answer to exactly this burden. Copilot gained multi-file editing inside the workspace; Claude Code gained direct terminal delegation; Codex receives a repository plus sandbox; Devin receives shell/editor/browser access. [10, 12, 26, 39]

H12 - tool-trace bias. Specialized tools automatically produce auditable development artifacts: diffs, branches, pull requests, test logs, agent sessions, and repository links. An ordinary ChatGPT user, by contrast, may have used a thousand prompts without the repository containing a single trace of that fact. Public software history therefore has a mechanical bias in favor of IDE/agent development. This is an inference from the differences among the systems and from the ways DevGPT/Programming-by-Chat can collect their data. [6, 9, 41]

K. Documentation problem

This is probably the core of the secondary research question.

A GitHub repository can reliably prove **what** was built, but almost never **how** the author arrived at each change. A ChatGPT conversation export can show the "how," but is usually private, has no stable public project ID, and may be distributed across dozens of separate chats. An app-store page proves deployment, but not code authorship. A Reddit post can describe the process, but cannot automatically verify the claims.

Strict chat-native development is therefore particularly ill-suited to traditional prior-art search. The relevant information is distributed across at least three archives:

conversation history -> external code/artifact history -> deployment/publication history.

Most search engines index at most one or two of these together.

The Circle Clicker post illustrates this almost perfectly. The title says "100% code"; only the body reveals that it took a year, that small chunks were necessary, that linked files became problematic, and that one task per chat window worked better; the comments provide the concrete Play Store link. Without all of those pieces, the same case could easily be classified as a simple "I asked ChatGPT to make a game" demo. [2, 3]

That makes **historiographic scarcity** a better term than simply "rarity" for the current state of knowledge.

Artifact Accumulation, Vibe Coding, and Agentic Development

L. Artifact accumulation

The most interesting property of the target category is probably not natural-language programming by itself, but the **length of the causal chain**.

The basic unit is:

```
intent -> generated artifact -> execute -> observe -> feedback -> modified artifact ->
execute again -> new requirement -> further modification
```

In a short experiment, this happens three or five times. In sustained chat-native development, the same mechanism is repeated dozens, hundreds, or potentially thousands of times while the external software artifact contains ever more state.

This creates what I propose in this report to call the:

> **artifact accumulation loop** - a sustained human-LLM feedback loop in which each conversational intervention builds on an external executable artifact that itself carries the cumulative project state.

This should explicitly be understood as a **proposed analytical term**, not as a claim that the literature already uses it in this way.

Existing research addresses almost all of the components separately. Ross et al. explicitly argue that conversation history and the artifact-under-development form additional context layers, and demonstrate extended multi-turn software conversations. IntelliExplain describes conversational code generation as a process in which users detect errors or mismatches and request refinement through corrective feedback. *Self-Refine* formalizes generation -> feedback -> refinement as an iterative structure, but lets the model provide its own feedback and does not study the human project loop. LLM-supported program-design work likewise describes software design as iterative navigation through a solution space. [16, 77-79]

What appears much less explicitly in the literature examined here is **the same ordinary-chat loop treated as a longitudinal project unit** across multiple sessions, files, deployments, and changing requirements. This is not evidence that nobody has previously theorized it; it is a concrete gap in the corpus found here.

The artifact also performs a crucial memory function. In strict chat-native development, there are in practice at least two state stores:

conversation state contains recent intentions, explanations, and earlier responses;

artifact state contains the actual cumulative truth: the current files, database, runtime behavior, assets, and deployment.

Historically, the human was the synchronization protocol between the two.

This explains why the coaststl post devotes so much attention to keeping files small, re-uploading code, using unique names, making backups, and returning to a working version after failed debugging. These are not merely programming tips; they are techniques for making a weak synchronization layer between LLM context and artifact state artificially reliable. [38]

Circle Clicker shows the same thing in simpler form: "small chunks," one task per chat, and problems once files became linked to one another. [2, 3]

This suggests a useful new HCI measure: **human synchronization burden**. The larger the project becomes, the more the human must know which parts of reality the chat does or does not still know. That burden does not necessarily increase linearly with lines of code; dependencies and cross-file state may make it closer to combinatorial.

This also helps explain why a relatively small app can take a year even when the creator does not write code personally. The bottleneck shifts from *typing code* to *keeping state coherent*.

M. Comparison with modern vibe coding and coding agents

The structural differences can be summarized as follows:

Dimension	Strict general-purpose chat-native	Vibe coding	Coding agent
Primary interface	General chat	Varies: chat, IDE, builder, agent	Agent/IDE/CLI
Natural language	Dominant	Dominant	Dominant
Is reading code necessary?	Not necessarily	Originally often minimal	Human can remain high-level
Who opens files?	Human supplies them/context	Varies	Agent
Who modifies files?	Human copies AI output/follows instructions	Varies	Agent
Who runs tests?	Human	Human or tool	Agent
Who sees browser/runtime?	Historically human	Varies	Increasingly agent
Loop controller	**Human**	Varies	**Agent harness + human at a higher level**
Persistence	Manual/conversational	Varies	Repository/project-aware
Autonomy	Low	Low to high	Medium/high
Required for this study	**Yes**	Subset only	No

Karpathy's original vibe-coding description is therefore culturally important but taxonomically insufficient. "I barely read the diffs, paste errors back, and proceed by feel" describes **human oversight and epistemic stance**, not where the development environment resides. [25, 27]

Modern agentic programming changes something more fundamental. An agent can itself have an observation/action loop. OpenAI's description of Codex literally shows an agent loop in which the model calls tools; surveys define agents through autonomy, task decomposition, tool interaction, and iterative self-correction. [7, 9]

A person in 2026 can therefore "type almost nothing but prompts" and still be doing the methodological opposite of a 2023 ordinary-ChatGPT builder.

For the 2023 builder, the human is:

```
prompter + file router + runtime operator + tester + error observer + rollback manager + deployment operator
```

With a modern coding agent, the human is more often:

```
goal setter + reviewer + escalation point
```

This is the fundamental **interface-versus-model** distinction that is lost in much public discussion of "AI coding."

The shift is also historically visible in product design. From November 2024, Windsurf described its editor as an IDE built around an integrated AI engine; Claude Code became a tool for delegating substantial engineering work from the terminal; Codex received cloud sandboxes and parallel tasks; Devin has a complete software-engineering environment; GitHub Copilot grew from 2021 autocomplete to multi-file editing and agentic workflows. [10-12, 22, 26, 39]

Ironically, this means sustained natural-language development probably became **far more common** after 2025, while the particular variant studied here - *ordinary general-purpose chat remains the primary environment* - may have become relatively narrower because better specialized alternatives became available.

Comparison Case, Uncertainty, and Conclusion

N. Comparison case: Kloojo / GPTGTA / MadeFromChat

Raynor Eissens, Kloojo, GPTGTA, and MadeFromChat were examined only after the independent corpus above had been built.

Raynor Eissens' own methodology page explicitly describes:

```
human intent -> GPT conversation -> artifact -> human testing -> feedback to GPT -> next iteration
```

The MadeFromChat methodology page describes an intended strict workflow in which GPT/ChatGPT is the exclusive AI coding collaborator and explicitly excludes Claude, Claude Code, Cursor agents, GitHub Copilot, Replit Agent, Devin, Windsurf agents, autonomous Codex workflows, and multi-agent coding. Ordinary infrastructure - JavaScript, Three.js, Node.js, WebSockets, servers, GitHub, APIs, and databases - is allowed. Version 1.1 adds an author clarification that this site-level statement cannot be applied without qualification to Kloojo's complete current code history: Kloojo's main 3D game was built through sustained ordinary ChatGPT conversation, but its later multiplayer implementation used Codex to operate on local project files from a ChatGPT-generated prompt/instructions. [80, 86]

This correction makes the **phase boundary explicit**. GPTGTA remains reported as ChatGPT-only; the MadeFromChat park artifact itself remains a claimed strict case; Kloojo must be classified temporally rather than as one uninterrupted strict workflow. These are still creator workflow claims and do not automatically become Grade A without a complete prompt-to-repository audit trail.

Comparison case	Public artifact	Methodological evidence	Assessment
GPTGTA	2D top-down browser city/crime sandbox with traffic, stealable vehicles, interiors, garages, rooftops, metro/chopper elements; public build and video/X traces. [80-83]	Creator describes sustained ChatGPT-only prompting and a start around mid-May 2026. No complete prompt log in the material examined.	**Strict, Grade B**
Kloojo	Browser-native 3D skating/graffiti/traversal game; public documentation mentions tricks, mobile controls, world design, collectibles, and multiplayer with synchronized players/state. [80, 84]	Creator clarification: the main 3D game was developed through sustained ordinary ChatGPT conversation. The later online multiplayer implementation used OpenAI Codex to work directly with local project files from a prompt/instructions produced in ChatGPT. [86]	Hybrid overall; substantial strict chat-native phase followed by Codex-assisted multiplayer. Grade B self-report.
MadeFromChat park	Public isometric/playable portfolio environment with 19 project attractions according to the site and gallery. [84, 85]	The MadeFromChat park/site artifact is described as GPT-only. Project attractions must be classified individually; Kloojo is not strict overall after its later Codex-assisted multiplayer phase. [80, 86]	Strict for the MadeFromChat park artifact itself at the claimed level; not a blanket classification of every contained project. Grade B.
Raynor Eissens methodology	Not a single artifact but an explicit development-method statement	The interface/tool boundary is described explicitly and is refined in v1.1 to distinguish strict projects or phases from later hybrid phases. [80, 86]	Methodologically useful classification statement; no separate "first" value assigned.

What this comparison **does not** support is a general first-mover statement. Older independent strict cases already exist in the corpus: Jerry Shi's ChatGPT-4 Sudoku build, Circle Clicker, and Big 2 Wild. [1-4]

The public GPTGTA post itself uses first-like language. For this study, that is a claim by the project creator, not an established historical conclusion. [83]

Nor can the available material responsibly establish that Kloojo is "more complex than all precedents." Its broad feature combination cannot be ranked scientifically without comparable codebase, runtime, conversation, and repository metrics. Version 1.1 also makes clear that Kloojo's online multiplayer layer must not be cited as evidence of a purely strict chat-native implementation, because that layer introduced Codex. [2, 80, 86]

What does appear relatively distinctive is the **explicitness with which the development constraint itself has become an object of documentation**. Most older cases document an artifact and mention incidentally that ChatGPT made it. MadeFromChat instead turns the development loop into a formal classification boundary - and the Kloojo correction demonstrates that the boundary can identify a real transition from strict chat-native work to a later agent-assisted phase. [80, 86]

That difference is historiographically more important than an unproven "first": it makes searchable a usage pattern that previously appeared mostly between the lines of "I built X with ChatGPT" posts.

For **kloojo.com** itself, no separate Grade-A process trail could be established from the indexed results accessible to this study. Version 1.1 therefore records the creator's post-publication clarification as Grade-B self-report: substantial ordinary-ChatGPT development preceded a later Codex-assisted multiplayer implementation. The exact prompt-to-file boundary remains unaudited in the public corpus. [86]

P. Remaining uncertainty

Open question	Why it does not follow from public evidence
How many people worldwide work in a strict chat-native way	No telemetry jointly measures interface, project persistence, code share, and agent use.
Average project duration	Conversation datasets identify sessions better than longitudinal artifact identities.
How many private projects exist	Unpublished ChatGPT histories are, by definition, not web-searchable.
How often people migrate from ChatGPT to Cursor/Codex/etc.	Many individual examples, no representative cohort data.
Exact share of AI code in most cases	Self-reporting; repositories do not reliably label AI authorship.
Whether deleted/private chat shares contain older strong precedents	Historical link rot and unarchived share links prevent full reconstruction.
Whether an existing term previously named the strict category exactly	Many semantically adjacent terms were found, but exhaustive proof of linguistic absence is impossible.
Exact audit trail for GPTGTA, MadeFromChat, and the phase boundary inside Kloojo	The creator now clarifies that Kloojo became hybrid when Codex was used for multiplayer, but the public corpus still lacks a complete prompt-to-commit/file audit trail for the comparison cases. [86]
Which strict case was historically "first"	Not established and probably not reliably recoverable from the current web archive.
How much complexity ordinary chat can ultimately carry	That ceiling moves with models, memory, file context, and product affordances.

Claims/evidence matrix

Claim	Strength	Main evidence
Strict ordinary-chat software development existed well before the current agent boom.	**High**	Jerry Shi, Circle Clicker, Big 2 Wild. [1, 2, 4]
The public archive is insufficient to quantify prevalence.	**High**	Massive ChatGPT usage versus selected GitHub/chat datasets without longitudinal project labels. [6, 15, 74]
Multi-turn coding conversation is not rare in itself.	**Medium-high**	CodeChat 68% multi-turn; another GitHub study around one third. [5, 6]
Context/state management formed a real ceiling.	**High**	Direct project experiences + technical motivation for agents. [2, 7, 38]
Migration to specialized tools is a plausible visibility drain.	**High qualitative**	Multiple ChatGPT -> Cursor/Codex cases plus rapid growth of specialized tooling. [10, 60, 61, 67]
Vibe coding is not a valid one-to-one synonym.	**High**	Original and later definitions impose no ordinary-chat/duration boundary. [23-25]
Interface/harness is at least as important as model choice.	**High**	Codex/agent literature describes tool loops and autonomy as system properties. [7, 9]
Artifact accumulation is conceptually present in existing research but not clearly named as its own longitudinal ordinary-chat category.	**Medium**	Ross, IntelliExplain, iterative refinement/program-design literature. [16, 77-79]
The comparison corpus contains both strict and hybrid histories; Kloojo has a substantial strict phase but is hybrid overall after Codex-assisted multiplayer.	High for the author-reported phase distinction; medium for independent verification	MadeFromChat methodology statement plus post-publication author clarification. [80, 86]
The comparison cases do not prove a historical "first."	**High**	Older independent strict precedents exist. [1, 2, 4]

Machine-readable research data

The full dataset contains the fields `year_started`, `artifact`, `complexity`, `primary_llm`, `primary_interface`, `natural_language_steering`, `human_manual_coding`, `specialized_coding_agent`, `sustained_iteration`, `duration`, `deployment`, `source_quality`, `evidence_grade`, `classification`, `exclusion_reason`, and `source_key`. Unknown values were preserved as `unknown` rather than inferred.

Full research dataset as JSON (`research_dataset.json`; companion research file)

All 32 examined cases as CSV (`all_examined_cases.csv`; companion research file)

The 29 independently examined blind cases as CSV (`independent_cases.csv`; companion research file)

Strict-match subset as CSV (`strict_matches.csv`; companion research file)

Excluded and near-match cases as CSV (`excluded_and_nearmatches.csv`; companion research file)

Source bibliography - selected core sources

Date	Source	Relevance
29 June 2021	GitHub, *Introducing GitHub Copilot: your AI pair programmer*	Pre-ChatGPT AI pair-programming baseline. [22]
30 Nov. 2022	OpenAI, ChatGPT launch	Starting point for modern public general-purpose chat. [30]
14 Feb. 2023	Ross et al., *The Programmer's Assistant*	Most important early academic formulation of conversational interaction grounded in code. [16]
14 March 2023	OpenAI, GPT-4	Model development in the first ChatGPT coding wave. [33]
2023	OpenAI, Code Interpreter/files rollout	Chat moves toward artifact manipulation/execution. [13, 31]
25 Sept. 2023	OpenAI, image/voice capabilities	Screenshot/visual feedback becomes possible. [32]
2023	Jerry Shi build log	Very direct evidence of conversational app development. [1]
May 2023	Sasha Petrov, Write Like Native	Early ordinary-ChatGPT full-cycle app development. [34]
April 2024	PublicParkBench, Circle Clicker	Strongest early year-long strict case. [2, 3]
2024	Big 2 Wild build report	Production mobile app, majority-AI implementation. [4]
June 2024	coaststl development report	Especially useful documentation of context/state friction. [38]
3 Oct. 2024	OpenAI, Canvas	Explicit move from "simple chat" toward longer coding/writing workspaces. [37]
Nov. 2024	Windsurf launch	AI-native IDE as a structurally different development loop. [11]
Dec. 2024	OpenAI, Projects	Shared chats/files/context and multi-session framing. [36, 43]
2 Feb. 2025	Andrej Karpathy, original vibe-coding post	Origin of the culturally dominant but broader term. [25]
Feb. 2025	Anthropic, Claude Code preview	Delegation to a terminal-native coding agent. [12]
16 May 2025	OpenAI, Codex	Cloud software-engineering agent with repository/sandbox. [10]
2025	CodeChat	Large real-world developer conversation corpus; 68% multi-turn. [5]
2025	Agentic-programming/code-agent surveys	Formal autonomy/tool-loop distinctions. [7, 8]
2026	*Programming by Chat*	Large IDE-native behavioral study; shows where current observability has shifted.

Date	Source	Relevance
		[41]
27 Feb. 2026	OpenAI, *Scaling AI for everyone*	>900M weekly ChatGPT users as prevalence denominator. [15]
June-Aug. 2026	MadeFromChat/Raynor project material	Comparison-case methodology and artifact documentation. [80, 82-84]
28 Aug. 2026	Eissens, post-publication Kloojo methodology clarification	Corrects Kloojo from fully strict to hybrid overall: strict ordinary-ChatGPT phase followed by Codex-assisted multiplayer. [86]

Suggested future research questions

Research question	Why decisive
Can voluntary ChatGPT exports be clustered into project identities across months?	Would allow session-level data to be converted into longitudinal development data.
How many prompts/turns does a strict chat-native production project contain on average?	Operationalizes "sustained" empirically.
How often does a user migrate from ordinary chat to an AI IDE/agent once the codebase reaches a certain size?	Direct test of H5.
Which metric predicts failure better: lines of code, file count, dependency graph, or amount of context that must be restored manually?	Tests the artifact/state ceiling.
How much manual coding is required before users no longer perceive their work as "AI-built"?	Helps calibrate self-reporting and taxonomy.
Are non-programmers or experienced programmers better at sustained artifact-state management through chat?	Tests whether coding knowledge replaces syntax knowledge or instead orchestration skill.
Can commit histories be linked to conversation timestamps without exposing private content?	Could make Grade-A provenance possible.
How does error recovery differ between ordinary chat and agentic coding with exactly the same models?	Isolates interface/harness from model capability.
How often are successful chat-native projects methodologically described at all?	Needed to quantify documentation bias.
Is a new category emerging in which ChatGPT Projects/Canvas/files provide so much persistence that "ordinary chat" and "development environment" converge without true autonomy?	Important because the interface boundary itself is evolving.

Q. Research conclusion

The historical record has **not adequately cataloged the phenomenon as a distinct practice**.

The field has seen its components. Academics study conversational programming, iterative refinement, co-creative programming, and later IDE-native chat. Practitioners discuss CHOP. From 2025 onward, vibe coding became the cultural umbrella. Meanwhile, the industry shifted rapidly toward coding agents. [16, 21, 25, 41]

But this category falls precisely between those frames.

It is not classical AI pair programming, because the human may write almost no implementation code.

It is not a coding agent, because the human performs the actual action/observation loop.

It is not a no-code builder, because ordinary source files, libraries, servers, and deployment infrastructure exist.

It is not necessarily vibe coding, because someone may follow the generated architecture and errors very systematically.

And it is more than "programming with ChatGPT," because the defining property is not **that** ChatGPT produces code, but that **the same conversationally governed causal chain keeps accumulating as the software artifact grows**.

The older strict cases make clear that this was practically possible before Cursor, Claude Code, and Codex. Circle Clicker even shows that a non-programmer could sustain the awkward version of this workflow for a year: decomposing prompts, compensating for context loss, managing files, testing, and continuing until a real store release existed. [2, 3]

At the same time, the near matches show why it did not automatically become the dominant way of working. As software becomes more complex, someone has to monitor project state, dependencies, tests, error logs, and versions. Some people therefore learn to code themselves; others add Copilot; others move to Cursor; and from 2025 onward a coding agent can take over much of that mechanical synchronization layer. [10, 12, 38, 61, 62]

This leads to the answer to the central question:

> **Sustained general-purpose chat-native software development is probably genuinely unusual as a stable, long-term endpoint of a development workflow - but the public record strongly exaggerates that unusualness.**

The method requires more human orchestration than modern agentic development, creating a natural pressure to add manual programming or migrate to specialized tooling. This is a real behavioral-selection effect.

But there is also a powerful documentation-selection effect: ordinary-chat development leaves almost no automatic public provenance, lacked a stable name for years, and is usually flattened retrospectively into "made with ChatGPT," "AI coding," or, after 2025, "vibe coding." Public searchability thereby loses exactly the dimensions - interface, duration, and causal continuity - that make the method distinctive.

The comparison with Kloojo, GPTGTA, and MadeFromChat does not turn this conclusion into a first-mover story. Older strict precedents exist. The comparison corpus remains useful because it documents the development loop explicitly: GPTGTA is reported as a strict ChatGPT-only case; Kloojo contains a substantial strict ordinary-chat phase and then crosses into a Codex-assisted multiplayer phase. That transition is itself consistent with the report's broader finding that state and file-access pressure can push sustained chat-native projects toward specialized tooling. [80, 86]

The more interesting historical question is therefore not:

"Who did this first?"

but:

"Why did a recognizable longitudinal way of working with the world's most widely used AI interface go so long without much of an analytical category of its own?"

Based on the available evidence, the most likely answer is that the individual actions were never rare. Prompting, receiving code, returning an error, and trying again quickly became commonplace. What is more unusual - and poorly documented - is **continuing to repeat that loop while one real software artifact accumulates causality, state, and complexity for months, without either the human or a specialized agent eventually taking over the development loop**.

References

Numbering follows first appearance in the report. Web sources were captured from the research record supplied for this publication edition.

[1] Case Study | From Idea to App Store in 12 Hours. javoft.com. Source

[2] I used ChatGPT to 100% code a (simple) Android game . reddit.com. Source

[3] Hey! I coded a full (simple) Android game using nothing but . reddit.com. Source

[4] I built a poker-like game in react native using chatGPT in 8 . reddit.com. Source

[5] Developer-LLM Conversations: An Empirical Study of Interactions and Generated Code Quality. arxiv.org. Source

[6] An Empirical Study on Developers' Shared Conversations . arxiv.org, 15 Mar 2024. Source

[7] A Survey on Code Generation with LLM-based Agents. arxiv.org, 31 Jul 2025. Source

[8] AI Agentic Programming: A Survey of Techniques . arxiv.org, 15 Sept 2025. Source

[9] Unrolling the Codex agent loop. openai.com, 23 Jan 2026. Source

[10] Introducing Codex. openai.com, 16 May 2025. Source

[11] Windsurf Launch. windsurf.com, 13 Nov 2024. Source

[12] Claude 3.7 Sonnet and Claude Code. anthropic.com. Source

[13] ChatGPT — Release Notes. help.openai.com. Source

[14] OpenAI raises $122 billion to accelerate the next phase of AI. openai.com, 31 Mar 2026. Source

[15] Scaling AI for everyone. openai.com, 27 Feb 2026. Source

[16] The Programmer's Assistant: Conversational Interaction with a Large Language Model for Software Development. arxiv.org. Source

[17] Conversational Interaction with a Large Language Model . dl.acm.org. Source

[18] ChatGPT Will Change Software Engineering — But Not in . betterprogramming.pub, 1670154594.0. Source

[19] Let's Talk: Conversational Software Development. thenewstack.io, 27 Oct 2023. Source

[20] Chat-Oriented Programming in Action. qconsf.com, 18 Nov 2024. Source

[21] Chat-oriented programming (CHOP) in action. sourcegraph.com, 27 Jul 2024. Source

[22] Introducing GitHub Copilot: your AI pair programmer. github.blog, 29 Jun 2021. Source

[23] Vibe Coding in Practice: Motivations, Challenges, and a . dl.acm.org. Source

[24] Octoverse: A new developer joins GitHub every second as . github.blog, 28 Oct 2025. Source

[25] Andrej Karpathy on X: "There's a new kind of coding I call " . x.com. Source

[26] Introducing Devin, the first AI software engineer. cognition.ai, 3 Dec 2024. Source

[27] Vibe coding: programming through conversation with . arxiv.org, 29 Jun 2025. Source

[28] barnesoir/chatgpt-vscode-plugin: A VS code . github.com, 10 Jul 2025. Source

[29] ChatGPT VSCode Plugin zh. marketplace.visualstudio.com, 8 Dec 2022. Source

[30] Introducing ChatGPT. openai.com. Source

[31] ChatGPT Code Interpreter beta rollout has begun. community.openai.com, 6 Jul 2023. Source

[32] ChatGPT can now see, hear, and speak. openai.com. Source

[33] GPT-4. openai.com, 14 Mar 2023. Source

[34] How I Built and Launched an App with ChatGPT-4. petrov.substack.com. Source

[35] Funding, growth, and the next frontier of AI coding agents. cognition.ai, 9 Aug 2025. Source

[36] How educators are using OpenAI's Projects feature. edunewsletter.openai.com, 19 Dec 2024. Source

[37] Introducing canvas. openai.com, 3 Oct 2024. Source

[38] My Experience Building an App with ChatGPT and ZERO . reddit.com. Source

[39] Multi-file editing, code review, custom instructions, and . github.blog, 29 Oct 2024. Source

[40] Memory and new controls for ChatGPT. openai.com, 13 Feb 2024. Source

[41] Programming by Chat: A Large-Scale Behavioral Analysis . arxiv.org. Source

[42] ChatGPT is now a partner for your most ambitious work. openai.com, 9 Jul 2026. Source

[43] ChatGPT Capabilities Overview. help.openai.com. Source

[44] Boosting Mixed-Initiative Co-Creativity in Game Design. dl.acm.org. Source

[45] ChatGPT built a web app for me — I can give up learning to . reddit.com. Source

[46] How I Used ChatGPT to Create a Game | by Aris Catsambas. betterprogramming.pub, 4 Mar 2023. Source

[47] How I used ChatGPT to exploit another tool which uses . medium.com. Source

[48] Adrian Hupka Hupka. github.com. Source

[49] ChatGPT 4: 25 messages every 3 hours for $23/month?. reddit.com. Source

[50] That was a good productive month coding with GPT-4 . reddit.com. Source

[51] adimango/slack-pii-bot. github.com. Source

[52] Will Ahmed - ai #whoop #chatgpt #culture #engineering. linkedin.com. Source

[53] Propdecks – Apps on Google Play. play.google.com. Source

[54] PropDecks — A New Way to Play Fantasy Football. propdecks.com. Source

[55] How to Actually Build MVPs When You Don't Code. reddit.com. Source

[56] Sorry Siri, I Built My Own Assistant | by Ann Jackson - Medium. annujackson.medium.com. Source

[57] AI Replaces Paid SaaS Tools by Letting You Build Your Own. podcasts.apple.com. Source

[58] AI Replaces Paid SaaS Tools by Letting You Build Your Own. youtube.com. Source

[59] How I built a full-stack SaaS app in 9 weeks as a Product . reddit.com, 1761678146.65. Source

[60] 52 years old, no coding background. Built something in 5 months. indiehackers.com. Source

[61] My laptop got stolen and was able to completely remake . reddit.com. Source

[62] I thought AI would build my app for me... Here's what . reddit.com. Source

[63] I Built a $10K-Looking AI App in ChatGPT-5 in 25 Minutes . natesnewsletter.substack.com. Source

[64] A Month of Chat-Oriented Programming. checkeagle.com, 12 Nov 2025. Source

[65] My month of chat-oriented programming with Claude Code. linkedin.com. Source

[66] Andrej Karpathy Vibe Coding. klover.ai, 12 Jun 2025. Source

[67] From Zero Coding Experience to Published Android Game . reddit.com. Source

[68] Kleine Unternehmen schaffen mehr mit ChatGPT. openai.com, 4 Feb 2026. Source

[69] I Open-Sourced My UFC Prediction Model, Weights, and . mcinerney.ai, 5 Jun 2026. Source

[70] How a Swift Beginner Completed an iOS App Using Only . note.com. Source

[71] How are people building apps with AI with no coding . reddit.com. Source

[72] News & Press. replit.com. Source

[73] Insights from Developer-ChatGPT Interactions on GitHub. arxiv.org, 6 May 2025. Source

[74] Insights from Developer-ChatGPT Interactions on GitHub. arxiv.org, 18 Feb 2026. Source

[75] WildChat: 1M ChatGPT Interaction Logs in the Wild . arxiv.org. Source

[76] Developer-LLM Conversations: An Empirical Study of . arxiv.org, 12 Sept 2025. Source

[77] Self-Refine: Iterative Refinement with Self-Feedback. arxiv.org. Source

[78] IntelliExplain: Enhancing Conversational Code Generation . arxiv.org, 8 Oct 2024. Source

[79] LLM-supported Exploration of the Program Design Space. arxiv.org, 10 Mar 2025. Source

[80] Raynor Eissens — GPT-Only Sustained Conversational . madefromchat.com. Source

[81] GPTGTA — GPT-native 2D GTA-style browser sandbox, version . madefromchat.com. Source

[82] The First Playable Iterative ChatGPT-only Built Top-Down GTA- . youtube.com. Source

[83] GPTGTA — The first playable iterative ChatGPT-only built . x.com, 27 Jun 2026. Source

[84] Made From Chat Gallery | GPT-Built Browser Games, Tools . madefromchat.com. Source

[85] Made From Chat. madefromchat.com. Source

[86] Eissens, Raynor. Post-publication author clarification on Kloojo development history: the core 3D game was developed through sustained ordinary ChatGPT conversation; the later online multiplayer implementation used OpenAI Codex to work directly with local project files from a prompt/instructions produced in ChatGPT. 28 Aug. 2026. Author statement.

End of report