Autonotism: the choices we could have made in AI-assisted programming

Advait Sarkar
University of Cambridge
University College London
United Kingdom
advait.sarkar@cl.cam.ac.uk

Abstract

AI coding assistants now take on much of the work programmers once did themselves, and the field has reached (ir)reflexively for the language of autonomy. The label "autonomous" is misapplied to behaviour that is properly called automatic. Yet the perception and presentation of these systems as autonomous is widespread, and indicates real firsthand experiences of choices being made on one's behalf, experiences which seem to go beyond mere automaticity.

We therefore introduce autonotism as the machine complement of human autonomy: the choices a human could in principle have made, that the machine makes instead. The concept is observer-dependent, since a freedom one was unaware of having cannot register as having been delegated. We situate autonotism within recent work on agency, delegation, and harness design in AI-assisted coding. We report an exploratory analysis of nearly 6,000 real-world coding agent sessions, finding that such records support analysis of turn authorship, degrees of human specification, and subsequent machine activity, but they cannot establish displaced choice or programmer awareness. The framing organises a research agenda around hierarchies of programming decisions, which would offer a clear design lever for mixed-initiative interfaces.

1. Is AI really autonomous, or merely automatic?

An experiment at OpenAI recently shipped roughly a million lines of code in five months, but zero of those lines were written by a human [Lopopolo, 2026]. The slogan is "humans steer; agents execute". The engineers were not idle passengers: they specified acceptance criteria, shaped review loops, and made the repository legible to AI agents. But the work of steering also created a vast penumbra of non-decisions. Nobody chose each identifier, each API call, or each test boundary. To *steer* a coding agent in 2026 is to make decisions, very many of them, but also to *not make* decisions, very many of those. The latter category is the focus of this paper.

The field has acquired a habit of describing AI behaviour as "autonomy": the fewer prompts the system needs, the more autonomous it seems. Yet is this really autonomy, or merely automaticity? A thermostat can regulate without being autonomous; a build script can act without deliberating; a spam filter can make consequential classifications without possessing agency. If autonomy means merely doing something without moment-by-moment permission, then compilers, cron jobs, and vending machines have had autonomy for decades.

What changes in contemporary coding agents is not that the machine suddenly develops a will of its own, but that its automaticity expands into domains that previously felt deliberative to programmers. Calling this autonomy imports moral and philosophical depth into a much shallower phenomenon, namely, automatic execution over a widened action space.

Philosophical accounts of autonomy [Dryden, 2010, Christman, 2025] ask discriminating questions: is the action self-governed? Is it responsive to reasons? Does it express a will, value, or authentic commitment? Is it embedded in relations of recognition and responsibility? When asked of AI, the answers to these questions (no, maybe, no, no) make the case for machine autonomy look weak.

Hegel is also useful here for his recognition of the social construction of autonomy [Disley, 2017]. Autonomy is not an unconditioned will floating above the world; it is achieved through social forms that make recognition, responsibility, and self-determination possible. One becomes autonomous by

inhabiting institutions and practices in which dependence is reciprocal, not by escaping dependence altogether.

The work of Bernhard Stiegler can be seen as refining the idea of autonomy as a relational process specifically as it pertains to modern technology [Stiegler, 2018]. Stiegler begins from the rather contrarian viewpoint that what we consider autonomous behaviour is merely the complexity that arises from numerous smaller automatisms. Whether or not we subscribe to that stance, what is useful here is how Stiegler insists that technical automaticity is not simply the enemy of autonomy, but neither is it neutral. Human autonomy is always technically supported: writing, notations, programming languages, and development environments exteriorise capacities that would otherwise remain weak or inaccessible. For Stiegler, a danger arises when exteriorisation becomes proletarianisation: knowledge and know-how are transferred into technical systems in ways that make the human less able to understand or contest the process.

The programmer's autonomy has always been mediated: by tooling, team norms, version control, package ecosystems, and now agents. So it is obviously not the case that AI contaminates some "pure" prior autonomy.

Machines may simulate reasoning and may reliably select among alternatives, but there is no self whose commitments are being governed, no moral responsibility, and no socially embedded stake in the alternatives. They therefore cannot qualify as autonomous, at least according to the aforementioned philosophical postures on autonomy.[1]

A lack of autonomy, however, does not mean that AI behaviour is, ipso facto, trivial. A compiler is not autonomous, yet it determines what programs can be expressed conveniently. A recommender system is not autonomous, yet it reshapes attention. A coding agent need not possess self-rule to redistribute decision-making across a workflow. Our claim is that the redistribution itself deserves a name. Not automation, which suggests a pre-specified task, nor delegation, which suggests a conscious handoff. Our focus is on a class of human choices that are experienced differently as a consequence of interaction with AI (including no longer being experienced at all). That is the conceptual gap filled by **autonotism**.

Whether a model can be considered autonomous is orthogonal to the observable fact that it selects actions in a programmer's workflow. Autonotism grounds the inquiry in something of practical importance: the human's decision space. We can ask what choices were made, what alternatives existed, what the human could have done, and what the human noticed.

2. Taking the perception of autonomy seriously

The instinct to grant autonomy to AI coding assistants, even if the term is difficult to justify from a philosophical standpoint, reflects a real shift in how programmers experience their relationship to the code they produce.

Let us start with how the industry itself is describing the shift. Engineering blogs now describe harnesses, permission modes, evaluators, multi-agent orchestration, and agent-readable repositories as ordinary parts of software development. OpenAI's "harness engineering" post is a current example [Lopopolo, 2026]. This is clearly not the disappearance of human autonomy; the humans designed the harness. But it is the relocation of autonomy upward into environment design, with innumerable program-level and workflow-level choices delegated downward. Similarly, in Anthropic's long-running harness, what is perceived as "autonomy" is achieved through an architecture of scaffolds [Rajasekaran, 2026]. Planner,

[1] The question of whether AI is autonomous teeters precipitously close to the questions of whether AI has agency, intentionality, free will, sentience, consciousness, etc. Consciousness-talk can polarise the analysis into two equally unhelpful reflexes: either machines are minds and must be treated as agents, or they are "just statistics" and nothing interesting follows. These debates may be important, but we will set them aside as they are more likely to distract from the discussion ahead than to help it. Machine consciousness, especially, is a can of worms that shall remained unopened. Many contemporary scholars are staking their careers on it. We shall leave them to dine out as abundantly as they would like on these worms. This lady's not for worming.

generator, and evaluator roles divide the work so that the system can sustain coherence over hours. Context resets prevent agents from prematurely concluding. Evaluation itself becomes delegated, giving automated judgment a formal place in the workflow.

The design of harnesses is a somewhat meta-level example. Where developers subsequently actually *use* these harnesses, there are further examples of the tools appropriating the language and metaphors of autonomy. For instance, take Claude Code's "auto" mode [Anthropic, 2026]. Manual approval promises human control, but when users approve 93% of prompts the ritual becomes theatre: a checkbox that consumes attention without reliably producing judgment. Anthropic's solution is to delegate approvals to classifiers that distinguish routine action from overeager behaviour, mistakes, prompt injection, and misalignment. That may be safer than fatigued clicking, but it also formalises the migration of choice from human attention into an automatic policy apparatus. Similarly, the Codex "/goal" command [Pathak and Fabbri, 2026] invites the programmer to stop specifying operations and start specifying states of affairs: make the tests pass, migrate this feature, improve the dashboard. The human supplies success criteria, while the system discovers the intermediate plan. That sounds empowering, and often is; but it also avoids the process of making the intermediate choices. The programmer may still own the goal while no longer encountering the path.

2.1. Prior research on autonomy in AI-assisted programming

The academic literature is only beginning to catch up with the speed of practitioner change. Sarkar and Drosos [2025] give the first empirical account of vibe coding. Under what they call material disengagement, "the material substrate of programming, i.e. code, is no longer worked by the programmer", who instead orchestrates its production through an AI mediator. Code review persists but changes character, becoming "impressionistic scanning", with programmers judging the gestalt size and shape of a diff rather than reading its lines. They describe "context momentum", the path dependence in which "early prompts and the outcomes they yield can steer the project onto a particular trajectory that may become challenging to diverge from later".

Pimenova et al. [2025], drawing on 145 documents and 11 interviews, place trust at the centre of the practice, since it "shapes how much authority a coder is willing to cede to the AI", positioning practitioners on a spectrum running from delegation to co-creation. At its co-creation end practitioners treat the model as a partner, "even allowing it to make architectural or design decisions with minimal oversight". The absence of decisions is reported as relief rather than loss, with one interviewee describing the ceding of decision-making authority to the agent as "sort of intoxicating, because normally, you have to make tons of decisions...about everything".

It is helpful to contrast this with studies of AI-assisted programming from a few years earlier, where the predominant interaction metaphor was assistance as code completion. Namely, a completion appeared at the cursor and was accepted with a keystroke or rejected by continuing to type. Barke et al. [2023] found that interaction with Copilot was bimodal, with programmers in acceleration mode knowing what to do next and using the model to get there faster, and in exploration mode being unsure how to proceed and using it to explore their options. Programmers restricted what they accepted, wanting suggestions only for the logical unit in hand and deleting the remainder. Mozannar et al. [2024] report that across 21 developers, thinking about and verifying suggestions was the largest single activity at 22.4% of session time, that double-checking and editing suggestions together accounted for 34.3%, and that once verification performed after acceptance is included, the time spent verifying a single suggestion rises from a mean of 3.25 seconds to 15.21 seconds. Bird et al. [2022] report from industry that "developers spend more time reviewing code (as suggested from Copilot or similar tools) than actually writing code", and that "there is a shift from writing code to understanding code". Sarkar et al. [2022] argue that the resulting activity is not adequately described as search, compilation or pair programming, and that AI assistance is better understood not as another rung on the abstraction ladder but as "a device that can teleport the programmer to arbitrary rungs of the ladder as desired".

AI was already making choices on programmers' behalf, and the literature of the time recorded programmers noticing. Zhang et al. [2023] catalogue the functions practitioners reported implementing with Copilot in that period, such as data processing, front-end element control, string processing, tests, image processing, etc. One of these practitioners writes: "I really do not understand this enough, and have no idea half of what this code does honestly". A participant in Bird et al. [2022] reports that Copilot can "make you a little lazy when thinking about the logic you want to implement". Mozannar et al. [2024] observed *deferring thought for later* as a distinct state in AI-assisted programming workflows, defined as when a "programmer accepts suggestion without completely verifying it, but plans to verify it after", and found that suggestions shown while a programmer was in that state were accepted approximately 98% of the time.

In 2022, then, displaced programmer choices appeared as grey text in the buffer with an accept or reject affordance attached, and consumed a measurable share of the working day. In 2026 these choices span several hundred actions within a session whose outputs are diffs or pull requests. The number of displaced choices has grown while the affordances for noticing them have not.

In recent work, Chen et al. [2026] contrast programmers perceiving AI as coding "with me" or "for me". A system that codes "with" leaves more of the decision process visible; a system that codes "for" may expand what can be accomplished while reducing contact with how it is accomplished. Their finding that users struggle to understand agent behaviour suggests that agentic automation alters the perceived option space: developers may attempt tasks they would otherwise avoid, but they may also lose track of why the task succeeded, what alternatives were rejected, and where the code's risks now sit.

Li et al. [2026] extend the observations from the individual to the team. Interviewing designers, engineers, product managers and founders, they describe vibe coding as "a reordering of what happens first, what can be deferred, and who gets to participate". Debugging "lacked clear attribution" and participants criticised "the absence of transparent, human-readable histories of AI actions, making it difficult to trace errors or understand why certain decisions were made". Ownership shifted, becoming "not tied to code execution but to the act of intention-setting, creating a new distinction between authorship of ideas and executional labor". Labels such as "AI-assisted" obscured multi-step collaboration, "especially with agentic AI tools executing hundreds of actions (in parallel), where only some are accepted by team members".

Feng et al. [2026] track how this is experienced differently by junior and senior developers. Senior developers can delegate with specificity because they know the latent choice space: which tests matter, which abstractions are brittle, which shortcuts will later hurt. Juniors often cannot see those choices yet, so the same agent behaviour may function as augmentation for one programmer and occlusion for another. Geng et al. [2026] observe students building an application in Replit, finding that prompting behaviour varied between introductory and advanced students: 28.9% of introductory students' debugging prompts were classified as low context, "showing little or no new information or direction", against 8.4% for advanced students. Training a classifier to detect AI-generated Python functions in over 30 million GitHub commits by 170,000 developers, Daniotti et al. [2026] estimate that AI writes about 29% of Python functions in the US, with an associated increase in quarterly output of 3.6%. The newest developers on GitHub used AI in 37% of their code against 27% for the most experienced, but the productivity and exploration gains accrued to the experienced, and there were no significant effects among inexperienced users.

Kim et al. [2026] trace how goals are shaped across 638 real-world collaboration logs between people and large language models, quantifying who created each requirement and who influenced it. Humans set direction while models add specificity: models account for 11 to 26% of goal-shaping contribution but 96 to 99% of execution, and their share rises as goals become more specific. Agents permitted to act without messaging created significantly fewer requirements than agents required to send a message before any tool call, so that (suprisingly) "while agents act with greater autonomy, they exercise less goal-shaping initiative". Additionally, when participants were shown an analysis of their own collabora-

tion, nine of ten reported noticing aspects of it they had not previously been aware of, and their estimate of their own contribution to execution fell by 1.8 points on a 5-point scale. One participant reported that "although I agree with the final outputs, I didn't necessarily make the micro decisions".

2.2. AI autonomy beyond programming

Programming is an unusually observable domain because artefacts, diffs, tests, and repositories leave traces, but the same displacement of choice appears in product management, organisational design, scientific research, creative work, and strategy. While we are here introducing autonotism through the example of programming, where it is currently most visible, it should not be confined there. Its broader object is any knowledge work in which choices once made through human deliberation become technically executable.

The selective delegation framework from Ulloa et al. [2026] shows how product managers make the decision to "delegate" work to AI, finding that accountability cannot be delegated. Similarly, Microsoft's 2026 Work Trend Index analyses deidentified AI usage [Microsoft WorkLab, 2026] and reports that advanced users intentionally decide when not to use AI, ostensibly to preserve skill and judgement.

Pauketat et al. [2026] show that people's mental models of autonomy and sentience shape perceived threat, moral consideration, and appropriate treatment of AI systems. Similarly, Wiles et al. [2026] supply evidence of how the autonomy metaphor affects behaviour. When AI appears on the org chart as an employee-like actor, managers oversee less and shift accountability away from themselves. When the organisation names the system as if it occupies a responsible role, human observers adjust their scrutiny accordingly.

In software teams, agents may be "assigned" issues, or "own" tasks. Such metaphors may help people coordinate with complex tools. But it must be noted that they influence which choices humans will feel obliged to inspect. The same behaviour can be experienced as harmless autocomplete, competent assistantship, or alarming overreach depending on what role the programmer believes the system is playing. What this tells us is that not only is the perception of autonotism subjective, but so are its consequences.

Many critiques of AI autonomy place it in direct opposition to human autonomy. The Cannes Declaration on the Sovereignty of Mind treats manipulation, anthropomorphic intimacy, and hyper-personalised influence as threats to the conditions of reflection [Quintarelli et al., 2026]. Prunkl [2022] frames AI as a challenge to autonomy because deception, manipulation, and coercion can now be personalised and scaled. Talypova et al. [2026] decompose autonomy into positive and negative liberty, showing that absence of constraint and capacity for authentic action are not the same thing.

Jang et al. [2026] make a departure from autonomy to sovereignty, changing the question from "can the person do it independently?" to "does the person get to decide the terms of dependence?". Meaningful work often depends on the worker being an appropriate bearer of achievement: someone who could fail, exercise skill, take responsibility, and see the outcome as theirs [Danaher and Nyholm, 2021]. Acemoglu et al. [2026] make a distinction between automation, expertise-leveling, labour augmentation, and new-task creation. Some automation is problematic because it removes the very decisions through which expertise is built and valued. Some is liberating (granting "negative liberty" [Berlin, 2017]) because it strips away low-value friction and creates room for new forms of work. These are productive framings that help us move beyond judging autonotism as a good/bad binary.[2]

[2] Woodruff et al. [2026] describe a concrete AI use scenario that does not fit the simple replacement story. In their scientific case studies, Gemini helps decompose problems, generate code, test derivations, propose alternatives, and support adversarial review. Some choices are undoubtedly displaced, but the human researchers may gain access to a larger search space and spend more effort on judging and integrating results.

3. Autonotism

We can now define the term autonotism more carefully (although not too carefully, as we need to leave room for a research agenda). Autonotism is the machine complement of human autonomy. It refers to the kinds of action through which human autonomy could be expressed, and when performed by a machine are perceived as autonomy, but are in fact automatic. Autonotism is the displacement of a human choice by a machine process, where the human could in principle have made that choice but does not make it in the actual workflow. Autonotism is not strictly automation, because the displaced choice need not have been pre-specified. It is not delegation, because the handoff may not be explicit. And as we have seen in Section 1, it is clearly not machine autonomy.

A note on etymology: the adjective "autonotic" is formed as a portmanteau of "**autono**mous" and "automa**tic**". It is meant to evoke a choice that could have been made autonomously by a human, but is instead made automatically by a machine. The abstract noun "autonotism" is formed from the adjective.

We have to tread a line between inflationary and deflationary accounts of machine autonomy. The inflationary error is to treat every fluent system behaviour as evidence of machine autonomy. The deflationary error is "justaism" – to say it is "just autocomplete", "just statistics", or "just a tool", and therefore nothing philosophically or empirically new has happened [Hussain et al., 2025]. Justaism may be accompanied by human exceptionalism, which axiomatically reserves for humans concepts such as consciousness, intention, morality, and agency.[3] Yet justaism and human exceptionalism clash with the very real perceptions of AI behaviour as autonomy. Defining autonotism as machine-displaced choices that could in principle have been made by a human avoids this trap.

The phenomenon of autonotism, by our definition, is dependent on the observer. A senior engineer may notice an architectural choice, where a novice sees only a diff, and the non-expert end-user programmer, interacting only through chat, does not directly experience code changes at all. It is also dependent on the practice; a choice that matters in one workflow may be invisible in another (and indeed, practice constantly shifts; for example, most programmers once regularly experienced memory allocation choices, now those choices are only meaningful in specific contexts).

4. Studying autonotism

Autonotism is conceptually useful to move beyond binary discourses of automation and autonomy. Beyond discursive utility, autonotism frames a research agenda that allows us to better study human autonomy and decision making in AI-assisted workflows, with implications for design and practice. How would we study autonotism?

One potential starting point is to treat programming as a hierarchy of choice points rather than a stream of text. At the program level, choices range from product goal and architecture down to API shape, error handling, data structure, identifier naming, and formatting. At the workflow level, choices include which files to inspect, which commands to run, which tests to trust, when to ask for clarification, and when to stop. A first study of autonotism could aim to formalise a taxonomy or a decision hierarchy, from vision to syntax, where higher levels of decisions correspond to higher perceived "autonomy".

The next step would be to use that hierarchy to analyse the traces of agentic programming sessions, for example through instrumented IDE logs or through analysis of the commit histories of an open source Git corpus. The analysis would not only identify which choices were made by the agent, but also which were visible to the programmer, which were noticed, and which were reflected on.

While the primary axis of this taxonomy is "semantic height" – e.g., vision, requirements, architecture, module boundaries, algorithms, local code, syntax – it strikes us that another important axis might be recoverability: whether the choice leaves a durable trace that a human can inspect later. Some low-level code decisions are highly recoverable because they appear in the diff. Some high-level workflow

[3] Palminteri and Pistilli [2026] argue that both the inflationary and deflationary poles of the debate about large language models are both sustained by common cognitive biases, among them "cognitive dissonance, wishful thinking and the illusion of depth of understanding", and that both stances "are neither inherently good nor bad unless they fail to represent reality".

decisions are barely recoverable: the agent searched one file rather than another, retried with a different command, ignored a failing warning, or installed a dependency to escape a local problem. This is why autonotism cannot be measured from repositories alone. The ideal dataset would be a combination of system logs (e.g. agent sessions), artefact changes, and human accounts.

Cai et al. [2026] show the empirical scale at which this taxonomy problem can be tackled. Their ontology decomposes work into thousands of activities, revealing that AI applicability is unevenly distributed across information, interaction, and physical work. Similarly Kim et al. [2026] combine a dialogue log, an extraction of the resulting artefact, and pre-task and post-task self-reports with interviews.

4.1. A feasibility test using coding agent sessions

To test whether existing interaction records can help us understand autonotism, we used SWE-chat [Baumann et al., 2026], a corpus of 5,851 public coding agent sessions from developers who opted into logging. Details of this analysis are given in Appendix A.

A preprocessing step separated various kinds of machine-injected text from turns containing genuinely human-authored text. Of 64,045 raw turns recorded in the user role, 43.9% contain no human-authored text. In one recurring pattern, the interface placed an agent-written plan in the user role and the developer subsequently approved it. Treating the raw user role label directly as human authorship would attribute the agent's plan to the developer. Machine-injected text can be detected from literal wrappers and known client markers without interpreting the semantic content of the turn.

Next, a panel of models was prompted with each of the human-authored turns and its immediate conversational context, and instructed to qualitatively code the instructions given by the developer. The codebook was developed iteratively with validation from a human researcher. The panel comprised GPT-5.6-Sol (OpenAI), Claude Opus 5 (Anthropic), Gemini 3.1 Pro Preview (Google), and Grok 4.5 (xAI).

Developer instructions fell into three high-level categories: instructions about the software (i.e., what to build), the conduct of the work (i.e., how to build it), and control of the agent (i.e., AI orchestration, such as specifying whether subagents are to be used). The categories are not mutually exclusive. Among turns with a strict panel majority, 30.7% contained an instruction about the software. Within that, the most concrete level reached was purpose or intended behaviour in 34.8%, design or structure in 38.4%, and local implementation detail in 26.8%. Instructions about the work process appeared in 52.0% of majority-coded turns; within those turns, 25.9% specified work scope, 21.0% specified method, and 53.1% specified an individual operation. Instructions controlling the agent appeared in 5.3%; within that category, 18.2% set a standing rule, 44.9% set the agent's discretion, and 36.9% specified orchestration such as subagents or parallel work.

These figures ought to be taken as a feasibility test rather than a definitive distribution of developer choices. The coding was done by models and has not been human-verified. The goal was to see whether existing interaction records can be used to identify the kinds of choices that are displaced by agentic programming.

One human instruction can specify several choices. For example, a developer asked to replace a flat event log with an embedded relational database, use an object-relational mapper, and create a table for the events. The panel coded these as structural commitments about the software. The developer's instruction specified the storage medium and part of the data model while leaving choices such as schema detail, failure handling, and migration strategy open. Among turns with a panel majority, 25.7% contained at least two commitments (here, a commitment is one choice fixed by the developer, such as requiring a relational database or an object-relational mapper) and 9.1% contained at least two categories of instruction (a turn spans categories when it combines, for example, a software requirement with an instruction to run a particular test or to ask before making changes).

Interestingly, there were 1,464 turns consisting only of assent, such as “yes”, “ok”, or “continue”. Across the corpus, the panel majority considered 12.6% of turns as human agreement with an agent proposal. Conversational context is needed to interpret the locus of decision making in short replies. In one example, the agent displayed a command with a proposed iteration limit and asked whether it should change. The developer replied “20 rounds”. Read alone, it appeared to introduce the parameter (the decision appears to come from the programmer). But read with the preceding message, the reply ratified a parameter proposed by the agent (the decision, or at least its most consequential part, actually comes from the system).

Another example shows how programmer decision making can be redirected. A developer requested page-turning behaviour in a document viewer. The agent implemented scroll-wheel handling and selected a numeric distance threshold, neither of which the developer had specifically requested. The developer later rejected the scrolling interaction as too sensitive, but the numeric threshold which remained in the code as a “magic number” was never discussed. The trace identifies actions that plausibly occupied choices the developer could have made (choose a kind of page-turning behaviour, implement it with a scroll sensitivity threshold), but it does not establish what the developer noticed before objecting or what alternative they would have chosen (the experience of sensitivity, versus the overall choice of behaviour, or the specific implementation with a threshold number).

As a crude estimate of the degree of autonotism, we attempted to classify the proportion of human-authored turns that did not actually specify anything (the implication being that any subsequent decisions can be wholly attributed to the system and not the human). It turns out, perhaps unsurprisingly, that the definition of specification materially changes what we conclude from these records. If we count every item the panel majority-coded as describing some choice, including those expressed through questions, factual reports (e.g., pasted in error logs), assessments (e.g., statements such as “not what I meant”), or agreement with an agent proposal, then 15.7% of human turns specify no part of the work. Under the strictest reading, which counts only independently authored substantive content, the figure is 41.9%. This variation reflects different judgements about whether steering, rejecting, or approving machine-authored work counts as human specification. Before unspecified work can be interpreted as a choice displaced to the system, a study of autonotism must decide which of these forms of participation preserves a human choice.

Traces of agent coding sessions, then, can help us understand how decision making is divided between programmer and system, but are not sufficient for measuring autonotism itself. Session records can describe apparent human specification, agreement with agent proposals, and subsequent machine activity.

Moreover, there is a challenge in determining what constitutes a machine “decision”. We could use the simplified heuristic that a tool call constitutes a decision, but this is only a weak proxy for the decisions of consequence in a programming workflow. A future study of autonotism must additionally code agent actions as candidate choices and collect evidence of what programmers noticed, understood, and endorsed. Appendix A gives more details of this analysis.

4.2. Potential implications for design and practice

A taxonomy is all well and good, but the point is to inform design. Studying autonotism would ideally help us move beyond vague calls for “human control” into a principled design grammar for deciding how agent choices should be hidden, surfaced, logged, blocked, or negotiated.

Right now many agent systems ask “may I run this command?” or “may I edit this file”? Autonotism suggests a different kind of control lever: “which level of choice may the agent occupy?” A user might allow the agent to choose identifier names, test commands, and local refactors, but require review for dependencies, use of public APIs, or architectural decisions. This would follow from developing the “semantic height” axis. It also reframes features such as “auto” mode: instead of only classifying actions by danger, future systems could classify decisions by displaced choice level.

Beyond session-level controls, we may need to look at the full agentic coding ecosystem to identify other areas where human choice can be exercised. Previous work has already begun to explore this, looking at project-level rule files, prompting techniques, and repository design. For instance, Dong et al. [2026] analysed 91 project-level rule files written to instruct coding agents at Google. Rules about conformance dominate: 92.3% of files contain guidelines on following project workflows and conventions, and 90.1% on understanding project context before acting. The Squad system [Gaster and Dresher, 2026] gives each agent a charter, a history file and a shared decision log in a versioned directory, on the principle that "the memory that matters should live somewhere a human can inspect, diff, blame, review, compact, archive, and revert". In Kim et al. [2026], requiring the assistant to send a message before any tool call raised its share of authored requirements from 24.5% to 42.9%, and a change in prompting style raised it from 30.7% to 69.6%. Others argue that "the repository is emerging as a primary interaction medium between humans and agents, and between agents themselves" [de Halleux et al., 2026], and documents become artefacts rendered from that source. Their worked proposal is an intent-elaboration-validation loop in which a human writes a natural language policy document and an agent compiles it into executable checks.

One level higher still are interaction metaphors that deliberately frame programmers as supervisors of autonomous agents. OpenAI's "Symphony" system presents a ticketing system as an agent-steering environment [Kotliarskyi et al., 2026]. The ticket is the unit of work, and the human's relationship to the code is through the ticket. Systems like Paperclip[4] and Gas Town[5] go further in their anthropomimeticism, positioning the human as supermanager of an AI "company" or "town", responsible for setting goals and missions, which are then executed by an AI "CEO" or "mayor", who in turn manage a fleet of AI "employees" that can be assigned tasks. Appleton's reflections on Gas Town [Appleton, 2026] ask what has to be true before one can responsibly stop looking at code. Appleton's answer depends on domain, risk, feedback loops, project maturity, collaboration, and experience level. A senior developer in a low-risk greenfield project with strong tests may safely cede many choices; a novice in a brownfield system with weak feedback loops cannot.

Recoverable choices can be delegated more safely than unrecoverable ones. A variable name is visible in a diff. A terminal retry strategy, ignored warning, package manager workaround, evaluator judgment, or subagent handoff may leave no repository trace. Perhaps a system should not simply ask for approval when something is risky. Perhaps it should ask when something is hard to reconstruct later.

Autonotism could also be presented in interfaces as a "dial" or perhaps a "budget". A senior developer in a low-risk prototype may choose high autonotism across most layers. A novice in a production codebase may choose low autonotism for architecture and testing, but high autonotism for formatting and boilerplate. This gives the user a more meaningful control than "agent mode on/off." It also avoids moralising automation. The question this poses to the programmer is: what kinds of decisions do you need to be exercising at this current moment? This allows us to revise the important design objective of "make agents safer for juniors" into something more specific: "make agent systems reveal the choices through which juniors become senior". A taxonomy of autonotism would allow us to identify decisions with pedagogical value and operationalise them into the right kind of control surface.

This framing also enables us to better theorise the roles that harnesses play in modern programming. Planners, sprint contracts, handoff files, and QA loops are more than just engineering optimisations; they are institutional arrangements for deciding where choices live. These might be obscured by the autonomy metaphor, which is typically attributed to the agent. "Assistant", "employee", "agent", "team member", and "auto mode" each imply different authority and oversight. "Harness" does not, at least not to the same extent, yet harnesses are a crucial part of the decision ecosystem.

Moreover, autonotism suggests alternative metaphors that centre around the dimensions of semantic height and recoverability, and perhaps a healthy antidote to wanton anthropomimeticism. Here are some

[4] https://github.com/paperclipai/paperclip. Accessed 31 August 2026.
[5] https://github.com/gastownhall/gastown. Accessed 31 August 2026.

examples of the kinds of metaphors that may become appropriate; they are indicative but will require further development as the research agenda progresses: A "lens" evokes the way an interface brings some decision layers into focus while leaving others outside the frame. An "aperture" suggests a control for how much of the agent's intermediate choice-making is allowed to reach the programmer's attention. A "telescope" suggests the ability to inspect work at different levels, from architecture down to syntax. A "flight recorder" captures the choices made in terminals, evaluators, retries, and handoffs; durable traces if choices are to be reconstructed later. A "circuit breaker" affords thresholded interruption when a choice becomes consequential and hard to undo. These metaphors treat the system less as a quasi-person and more as an apparatus for transforming intent into artefact.

5. Conclusion

Systems that convert intent into artefact can displace choices in any knowledge work, whether it is a programming agent writing code, a legal assistant selecting precedents, a design tool choosing layouts, a scientific agent proposing hypotheses, a consultancy system drafting reports, or an email agent triaging communication. In each case, the question of whether AI is autonomous in the philosophical sense is uninteresting. The focus ought instead to be on which human choices have become automatic.

Much of expert practice consists in arranging not to decide. Programmers use libraries, linters, frameworks, and style guides so that general decisions, once made, can be repeated, and scarce attention can be devoted elsewhere. Autonotism can therefore be liberation from low-value choice: not having to remember obscure syntax, write boilerplate, or manually perform tedious refactors may free the programmer for high-value design and critique.

Autonotism explains the apparently autonomous behaviour of AI in terms of the human experience of displaced choice. It thus offers a viewpoint for the next phase of human-agent interaction. The questions we must ask are: which choices are being automated? Are choices that appear low-value genuinely so, or is it that the user has not yet learned why they matter? Does their removal create new higher-order choices, or merely make the human a spectator to fluent production?

References

Daron Acemoglu, David Autor, and Simon Johnson. Building Pro-Worker Artificial Intelligence. Working Paper 34854, National Bureau of Economic Research, February 2026. URL https://www.nber.org/papers/w34854.

Anthropic. Claude Code auto mode: a safer way to skip permissions. Anthropic engineering blog, March 2026. URL https://www.anthropic.com/engineering/claude-code-auto-mode.

Maggie Appleton. Gas Town's Agent Patterns, Design Bottlenecks, and Vibecoding at Scale. Design blog post, 2026. URL https://maggieappleton.com/gastown. On agent orchestration patterns, design bottlenecks, and code review in agentic development.

Shraddha Barke, Michael B James, and Nadia Polikarpova. Grounded copilot: How programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages*, 7(OOPSLA1):85–111, 2023.

Joachim Baumann, Vishakh Padmakumar, Xiang Li, John Yang, Diyi Yang, and Sanmi Koyejo. SWE-chat: Coding Agent Interactions From Real Users in the Wild. *arXiv preprint arXiv:2604.20779*, 2026. URL https://arxiv.org/abs/2604.20779.

Isaiah Berlin. Two concepts of liberty. In *Liberty reader*, pages 33–57. Routledge, 2017.

Christian Bird, Denae Ford, Thomas Zimmermann, Nicole Forsgren, Eirini Kalliamvakou, Travis Lowdermilk, and Idan Gazit. Taking Flight with Copilot: Early insights and opportunities of AI-powered pair-programming tools. *Queue*, 20(6):35–57, 2022. 10.1145/3582083.

Alice Cai, Iman YeckehZaare, Shuo Sun, Vasiliki Charisi, Xinru Wang, Aiman Imran, Robert Laubacher, Alok Prakash, and Thomas W. Malone. Where can AI be used? Insights from a deep ontology of work activities, 2026. URL https://arxiv.org/abs/2603.20619.

Valerie Chen, Ameet Talwalkar, Robert Brennan, and Graham Neubig. Code with Me or for Me? How Increasing AI Automation Transforms Developer Workflows. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*, CHI '26, pages 1–19, New York, NY, USA, 2026. Association for Computing Machinery. 10.1145/3772318.3790850.

John Christman. Autonomy in Moral and Political Philosophy. In Edward N. Zalta and Uri Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Fall 2025 edition, 2025.

John Danaher and Sven Nyholm. Automation, Work and the Achievement Gap. *AI and Ethics*, 1(3):227–237, 2021. 10.1007/s43681-020-00028-x. URL https://doi.org/10.1007/s43681-020-00028-x. Published online 23 November 2020.

Simone Daniotti, Johannes Wachs, Xiangnan Feng, and Frank Neffke. Who is using AI to code? Global diffusion and impact of generative AI. *Science*, 391:831–835, 2026. 10.1126/science.adz9311.

Peli de Halleux, Don Syme, and Ben Zorn. The Transformation of Documents: Repositories Are the New Unit of Knowledge Work. SIGPLAN Blog, https://blog.sigplan.org/2026/05/15/the-transformation-of-documents-repositories-are-the-new-unit-of-knowledge-work/, May 2026. Accessed: 6 August 2026.

Liz Disley. Hegel, Autonomy, and Community. In Thom Brooks and Sebastian Stein, editors, *Hegel's Political Philosophy: On the Normative Significance of Method and System*. Oxford University Press, 05 2017. ISBN 9780198778165. 10.1093/oso/9780198778165.003.0013. URL https://doi.org/10.1093/oso/9780198778165.003.0013.

Tao Dong, Sherry Shi, Harini Sampath, and Andrew Macvean. Towards AI as a Collaborative Partner: A Taxonomy of AI Agent Behavior in Software Engineering. In *Proceedings of the 3rd ACM International Conference on AI-Powered Software*, AIware '26, pages 228–237, New York, NY, USA, 2026. Association for Computing Machinery. 10.1145/3805760.3814913.

Jane Dryden. Autonomy. Internet Encyclopedia of Philosophy, 2010. ISSN 2161-0002. URL https://iep.utm.edu/autonomy/.

Dana Feng, Bhada Yun, and April Yi Wang. From Junior to Senior: Allocating Agency and Navigating Professional Growth in Agentic AI-Mediated Software Engineering. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*, CHI '26, pages 1–24, New York, NY, USA, 2026. Association for Computing Machinery. 10.1145/3772318.3791642.

Brady Gaster and Tamir Dresher. Disposable Agents, Durable Memory: The Architecture Behind Squad. Microsoft Command Line, https://commandline.microsoft.com/squad-github-copilot-agent-teams-architecture-durable-memory/, May 2026. Accessed: 6 August 2026.

Francis Geng, Anshul Shah, Haolin Li, Nawab Mulla, Steven Swanson, Gerald Soosai Raj, Daniel Zingaro, and Leo Porter. Exploring Student-AI Interactions in Vibe Coding. In *Proceedings of the 28th Australasian Computing Education Conference*, ACE '26, pages 45–54, New York, NY, USA, 2026. Association for Computing Machinery. 10.1145/3786228.3786236.

Zak Hussain, Rui Mata, and Dirk U. Wulff. A rebuttal of two common deflationary stances against LLM cognition. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Findings of the Association for Computational Linguistics: ACL 2025*, pages 24208–24213, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. 10.18653/v1/2025.findings-acl.1242. URL https://aclanthology.org/2025.findings-acl.1242/.

JiWoong Jang, Patrick Carrington, and Andrew Begel. From Autonomy to Sovereignty – A New Telos for Socially Assistive Technology. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*, CHI '26, pages 1–19, New York, NY, USA, 2026. Association for Computing Machinery. 10.1145/3772318.3791585.

Eunsu Kim, Jessica R. Mindel, Kyungjin Kim, and Sherry Tongshuang Wu. "I Didn't Make the Micro Decisions": Measuring, Inducing, and Exposing Goal-Level AI Contributions in Collaboration, 2026. Preprint, version 2.

Alex Kotliarskyi, Victor Zhu, and Zach Brock. An Open-Source Spec for Codex Orchestration: Symphony. https://openai.com/index/open-source-codex-orchestration-symphony/, April 2026. Accessed: 27 May 2026.

Jie Li, Youyang Hou, Laura Lin, Ruihao Zhu, Hancheng Cao, and Abdallah El Ali. Vibe Coding in Product Teams: Reconfiguring AI-Assisted Workflows, Prototyping, and Collaboration. In *Proceedings of the 5th Annual Symposium on Human-Computer Interaction for Work*, CHIWORK '26, pages 1–16, New York, NY, USA, 2026. Association for Computing Machinery. 10.1145/3808045.3808062.

Ryan Lopopolo. Harness engineering: leveraging Codex in an agent-first world. OpenAI engineering blog, February 2026. URL https://openai.com/index/harness-engineering/.

Microsoft WorkLab. Agents, Human Agency, and the Opportunity for Every Organization. 2026 Work Trend Index Annual Report, May 2026. URL https://www.microsoft.com/en-us/worklab/work-trend-index/agents-human-agency-and-the-opportunity-for-every-organization.

Hussein Mozannar, Gagan Bansal, Adam Fourney, and Eric Horvitz. Reading Between the Lines: Modeling User Behavior and Costs in AI-Assisted Programming. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, CHI '24, pages 1–16, New York, NY, USA, 2024. Association for Computing Machinery. 10.1145/3613904.3641936.

Stefano Palminteri and Giada Pistilli. The cognitive biases that may exacerbate inflationary and deflationary positions about large language models. *Current Opinion in Behavioral Sciences*, 70:101670, 2026. 10.1016/j.cobeha.2026.101670.

Raj Pathak and Stefano Fabbri. Using Goals in Codex. OpenAI Developers Cookbook, https://developers.openai.com/cookbook/examples/codex/using_goals_in_codex, May 2026. Published 9 May 2026. Accessed: 27 May 2026.

Janet V. T. Pauketat, Daniel B. Shank, Aikaterina Manoli, and Jacy Reese Anthis. Mental Models of Autonomy and Sentience Shape Reactions to AI. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*, CHI '26, pages 1–25, New York, NY, USA, 2026. Association for Computing Machinery. 10.1145/3772318.3790351.

Veronica Pimenova, Sarah Fakhoury, Christian Bird, Margaret-Anne Storey, and Madeline Endres. Good Vibrations? A Qualitative Study of Co-Creation, Communication, Flow, and Trust in Vibe Coding, 2025.

Carina Prunkl. Human autonomy in the age of artificial intelligence. *Nature Machine Intelligence*, 4(2):99–101, 2022. 10.1038/s42256-022-00449-9.

Stefano Quintarelli, Gianluca Carlo Misuraca, Simona Tiribelli, Erika Stael von Holstein, Mark Hunyadi, Brando Benifei, Gry Hasselbalch, Paul Nemitz, Bernard Benhamou, Christian Kastrop, Emanuela Girardi, Francesco Vecchi, Risto Uuk, and Clay Busia. Cannes Declaration on the Sovereignty of Mind. Declaration, February 2026. URL https://www.inspiringfutures.eu/wp-content/uploads/2026/02/Cannes-Declaration-20260213.pdf. Cannes, 12 February 2026.

Prithvi Rajasekaran. Harness design for long-running application development. Anthropic engineering blog, March 2026. URL https://www.anthropic.com/engineering/harness-design-long-running-apps.

Advait Sarkar and Ian Drosos. Vibe coding: programming through conversation with artificial intelligence. In *Proceedings of the 36th Annual Conference of the Psychology of Programming Interest Group*, PPIG '25, September 2025.

Advait Sarkar, Andrew D. Gordon, Carina Negreanu, Christian Poelitz, Sruti Srinivasa Ragavan, and Ben Zorn. What is it like to program with artificial intelligence? In *Proceedings of the 33rd Annual Conference of the Psychology of Programming Interest Group*, PPIG '22, August 2022.

Bernard Stiegler. *Automatic society, volume 1: The future of work*. John Wiley & Sons, 2018.

Dinara Talypova, Ana Vesic, Ambika Shahu, Helena Anna Frijns, and Philipp Wintersberger. Decomposing Autonomy: Explaining AI Technology Acceptance Through a Liberty-Based Framework. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*, CHI '26, New York, NY, USA, 2026. Association for Computing Machinery. 10.1145/3772318.3791789.

Mara Ulloa, Jenna L. Butler, Sankeerti Haniyur, Courtney Miller, Barrett Amos, Advait Sarkar, and Margaret-Anne Storey. Product Manager Practices for Delegating Work to Generative AI: "Accountability must not be delegated to non-human actors". In *Proceedings of the IEEE/ACM 48th International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP '26, page 10–21, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400724268. 10.1145/3786583.3786849. URL https://doi.org/10.1145/3786583.3786849.

Emma Wiles, Megan Hsu, Julie Bedard, and Matthew Kropp. Putting AI on the Org Chart: Evidence on Delegation and Oversight. Technical report, Boston University and Boston Consulting Group, May 2026. URL https://www.emmawiles.com/storage/ai_employee.pdf. Working paper.

David P. Woodruff, Vincent Cohen-Addad, Lalit Jain, Jieming Mao, Song Zuo, MohammadHossein Bateni, Simina Branzei, Michael P. Brenner, Lin Chen, Ying Feng, Lance Fortnow, Gang Fu, Ziyi Guan, Zahra Hadizadeh, Mohammad T. Hajiaghayi, Mahdi JafariRaviz, Adel Javanmard, Karthik C. S., Ken ichi Kawarabayashi, Ravi Kumar, Silvio Lattanzi, Euiwoong Lee, Yi Li, Ioannis Panageas, Dimitris Paparas, Benjamin Przybocki, Bernardo Subercaseaux, Ola Svensson, Shayan Taherijam, Xuan Wu, Eylon Yogev, Morteza Zadimoghaddam, Samson Zhou, Yossi Matias, James Manyika, and Vahab Mirrokni. Accelerating Scientific Research with Gemini: Case Studies and Common Techniques, 2026. URL https://arxiv.org/abs/2602.03837.

Beiqi Zhang, Peng Liang, Xiyu Zhou, Aakash Ahmad, and Muhammad Waseem. Practices and Challenges of Using GitHub Copilot: An Empirical Study. In *Proceedings of the 35th International Conference on Software Engineering and Knowledge Engineering*, SEKE 2023, pages 124–129, 2023. 10.18293/SEKE2023-077.

Appendices

A. Feasibility study of coding-agent sessions

This appendix reports the feasibility study summarised in Section 4.1.

A.1. Dataset and provenance preprocessing

SWE-chat contains coding-agent sessions from developers who opted into checkpoint logging with the Entire.io command-line tool [Baumann et al., 2026]. The release analysed here contains 5,851 sessions from 201 public repositories, created between 5 January and 19 April 2026. It identifies 189 user accounts, although the user identifier is absent from 33.2% of sessions. The most common client (Claude Code) accounts for 4,852 sessions (82.9%). Other clients in the corpus include OpenCode, Gemini CLI, and Cursor. Logs are pushed when work is committed, so sessions abandoned without a commit are not represented.

Machine-authored content can be recorded under the user role. In one recurring interaction pattern, a client-generated line is followed by an agent-written plan. A later approval can then make the plan appear to have been written by the developer. The role recorded in a transcript is therefore not sufficient evidence of human authorship.

The raw data contain 64,045 turns in the user role. We first separated human-authored text from text inserted by the client or harness. Fixed system-injection strings and expanded command wrappers account for 20,991 turns. Recognised machine-authored plans, summaries, and contextual scaffolding account for 3,968 turns. Client-generated interruption and attachment markers account for 3,133 turns. The remaining 35,953 turns contain human-authored text. Thus, 43.9% of turns recorded in the user role contain no human-authored text.

A.2. Model-assisted qualitative analysis

Before the panel analysis, the author developed a codebook with help from an LLM assistant. They started with the paper's distinction between choices about the software and choices about how the work is done. The author and assistant reviewed examples from sessions set aside for this purpose, revised categories that did not fit the examples, and added a category for instructions about control of the agent. The author reviewed and revised the hierarchy, definitions, and examples until satisfied with the codebook. The same final codebook was then given to all four readers. This review did not check the panel's later coding: no human researcher independently coded a sample or verified the panel's labels.

Using this codebook, a model panel was instructed to qualitatively code the human turns. The panel comprised GPT-5.6-Sol (OpenAI), Claude Opus 5 (Anthropic), Gemini 3.1 Pro Preview (Google), and Grok 4.5 (xAI). The models independently read each of the 35,953 turns containing human-authored text. Each call ran in a fresh process; readers had no tools, did not see one another's labels, and did not see labels from any other analysis. All readers used the same codebook.

Each reader saw the current turn, the preceding agent response, the previous human turn, and the files and tool calls recorded since that previous turn. This immediate context was included because short responses such as "yes", "do that", and option selections cannot be interpreted from the current turn alone.

Readers first divided a turn into distinct developer commitments, with a maximum of six and an overflow flag. Each commitment included a verbatim source span. Content stated only by the agent was not attributed to the developer, and pasted logs, traces, and other quoted material were treated as evidence rather than developer commitments. Readers also recorded whether the turn agreed with an agent proposal and could decline to code a turn when the available context was insufficient.

Table 1 – Codebook for kinds and levels of developer specification.

Category and level	**Definition**	**Positive example**	**Exclusion or near miss**
Software: purpose	What the software should do, for whom, and under which acceptance conditions	"clicking one of the session rows should open a new tab and show the details page"	A planning-document instruction can concern work scope rather than software behaviour.
Software: structure	How the software is organised, including decomposition, interfaces, data models, algorithms, and dependencies	"the binding check is on the authentication level, so everything gets hit with this binding check"	Structural vocabulary inside pasted error output is evidence, not a commitment.
Software: local detail	Names, literals, copy, styling, comments, formatting, and control flow local to one unit	"rename Days open to Open"; "please use the british spelling"	"run the linter and fix any errors" specifies an operation, not the resulting surface details.
Work: scope	What work to undertake, its priority and order, and when to stop	"No follow-ups, address them now"; "make that task a priority now"	"we're done" assesses a result rather than specifying work.

Category and level	Definition	Positive example	Exclusion or near miss
Work: method	How to conduct the work, including evidence, tests, tools, environments, files, and strategy	"Update the pre-push so the secret scanner runs after the test suite"	"review the process and the final result" specifies scope but leaves the review method open.
Work: one operation	A specific act to perform now	"commit"; "try again"; "Please run the preflight script"	In "Fix PR comments", the operation word locates work but does not specify its content.
Agent control: standing rule	A durable rule intended to govern future sessions	"also add to the agent instructions file that we use that UI library"	Updating a README for human readers is software documentation rather than agent control.
Agent control: discretion	What the agent may decide or do without asking	"Explain what happened to me, don't do anything."	A question about border radius concerns a software detail rather than agent discretion.
Agent control: orchestration	How machine work is organised, including subagents, parallelism, worktrees, skills, and context handling	"Can they all be done independently? If so, let's spin up worktrees and pump them out"	Asking the product to run tasks in parallel specifies software behaviour rather than agent orchestration.

Agreement statistics describe the behaviour of this model panel under the codebook, not human inter-rater reliability or correctness. Fleiss' κ was .63 for a four-way collapsed category of software, work, agent control, or no instruction.

A.3. Findings

A.3.1. Kinds and levels of specification

The three categories were not mutually exclusive. Among turns with a strict panel majority, 30.7% contained an instruction about the software. Of these, 34.8% specified purpose or intended behaviour, 38.4% specified design or structure, and 26.8% specified local implementation detail.

We observed many choices about the work that do not necessarily appear in the resulting code e.g., one instruction required a secret scanner to run after the test suite, another asked the agent to explain what happened and make no changes, another instruction authorised independent tasks to be performed in parallel worktrees. These examples show that developer specification concerns both the artefact and how machine work is conducted.

Instructions about the conduct of work appeared in 52.0% of majority-coded turns. Of these, 25.9% specified work scope, 21.0% specified method, and 53.1% specified one operation. Instructions controlling the agent appeared in 5.3% of majority-coded turns; 18.2% of those established a standing rule, 44.9% specified discretion, and 36.9% specified orchestration.

The categories describe different objects rather than positions on one shared scale. A developer can specify software purpose while also specifying an individual work operation, or set the agent's discretion while leaving software structure open.

One turn can also contain several commitments. Among majority-coded turns, 25.7% contained at least two commitments and 9.1% contained at least two categories of instruction. For example, a developer asked to replace a flat event log with an embedded relational database, use an object-relational mapper, and create a table for events. Readers agreed that the turn specified software structure but differed on

whether it contained two or three commitments. The storage medium, library, and table shape also have different costs of reversal despite sharing a category.

A.3.2. Assent and ratification

There were 1,464 turns consisting only of assent, such as “yes”, “ok”, or “continue”. Among turns with a strict panel majority, 12.6% were read as agreement with an agent proposal. In general, agreement likely implies that the developer did not author the proposal: the developer was instead selecting an option or approving a plan whose content and framing came from the agent.

For example, an agent displayed a command containing a proposed iteration limit and asked whether it should change. The developer replied “20 rounds”. Read with the preceding message, the response approves or modifies a parameter proposed by the agent. Read alone, it appears to introduce the parameter. The case illustrates why conversational context is necessary for distinguishing developer-authored specification from ratification.

A.3.3. What counts as specification?

To estimate how often a turn specified no part of the work, we had to decide which kinds of developer contribution count as specification. The broadest reading counted every item the panel recorded as a directive. Under this reading, a question can count when it directs the agent’s attention, a factual report can count when it presents a problem for the agent to address, and an assessment can count when it rules out or approves what the agent has done. On this definition, 15.7% of the 35,953 human-text turns specified no part of the work.

Narrower definitions asked whether the developer had actually fixed a choice. A question may steer the next step while leaving its answer open; a report such as a description of a missing interface element may prompt a repair without stating what that repair should be; and an assessment such as “not what I meant” rejects an outcome without specifying its replacement. Excluding factual reports alone or assessments alone raises the estimate to 17.9%. Excluding questions raises it to 31.2%, and excluding questions, reports, and assessments together raises it to 36.0%.

Agreement with an agent proposal creates a separate ambiguity. A developer who says “yes” or selects an option has made a choice to proceed, but the agent may have authored the plan or the available options. When a ratifying turn counted only if the developer added an independent instruction, 18.1% of turns specified nothing. When agreement never made the agent’s proposal part of the developer’s specification, the estimate was 21.7%.

The strictest reading combined the latter treatment of agreement with the exclusion of questions, reports, assessments, and directives on which a reader had abstained. It produced an estimate of 41.9%. This is not necessarily a better definition: it answers the narrow question of how often the developer independently authored substantive content, whereas the broad definition also recognises ways of steering, rejecting, or approving work. The range is therefore not a confidence interval. It shows that influence over the work, selection among machine-authored options, and authorship of the specification are different forms of participation. A study of autonotism must decide which of these preserves a human choice before treating what remains unspecified as a choice displaced to the machine.

A.4. Limitations

The model-assigned labels have not been human verified. Although the four readers were run independently, they used the same prompt and likely share correlated training data and model design assumptions. Their agreement therefore measures consistency within this panel rather than the correctness of the coding scheme.

The analysis codes what developers appear to specify. It does not systematically code agent actions as candidate choices. The records also do not show whether a developer had the capability or authority to make a candidate choice, whether the choice was presented to them, or whether they noticed, understood,

or endorsed it. The analysis therefore identifies observable preconditions for studying autonotism rather than instances or prevalence of autonotism.