# Agent-Driven Development

How to Stay in Control
While Delegating Development
to AI Agents

Code is cheap. Show me the spec.

Alex Gusev

# Agent-Driven Development

## How to Stay in Control While Delegating Development to AI Agents

Alex Gusev

This book is available at https://leanpub.com/adsm-en

This version was published on 2026-04-22


This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.



This English edition was translated from Russian with the help of AI and reviewed by the author.

# Contents

# Introduction

Large language models (LLMs) have already entered development and are rapidly changing how it is practiced. They help write code, accelerate implementation, and make it possible to delegate an ever larger share of engineering work to agents. At the same time, a new challenge is emerging. When AI agents are used directly, the result often turns out to be unstable, weakly reproducible, and difficult to keep manageable. An agent can significantly accelerate work. But execution speed alone does not define a mode of development that remains stable over the long term.

In my view, this is connected to the fact that modern development still relies heavily on the `code-first` model. Code is treated as the main source of truth and the usual center of management for system development. Yet a human specialist and a language model read code differently. For a developer, code crystallizes professional experience, architectural understanding, and knowledge of the system's constraints. For the model, code is also meaningful material, but it does not automatically activate the same engineering background. On the contrary, the model relates code to a much broader field of possible interpretations. A human usually reads code differently because they rely on more concrete, practically formed experience. Therefore, when implementation is delegated to an agent, the project and semantic context must be set explicitly. A human often reconstructs this context from the code on their own, drawing on their professional specialization. This is exactly why documentation, formulations, and the structure of context become more important as instruments for managing agent behavior and preserving reproducibility.

This leads to the central practical question: how should development with AI agents be organized so that the result remains predictable, manageable, and suitable for further development? As work with agents spreads, it becomes increasingly important not only to obtain individual solutions from the model, but also to build a stable process for working with it. This book examines one possible answer: the explicit setting of meaning, constraints, and implementation conditions in the textual context with which the agent works.

I call this approach ADSM (`Agent Driven Software Management`). It grew

out of my practical experience working with the Codex agent while developing packages for the Tequila Framework platform. In that work, code is systematically created, refined, and extended with the agent's participation, while the principles of interaction themselves gradually take on a more explicit and coherent form. ADSM therefore rests on accumulated practical experience, and the results of this joint work are open and publicly available in GitHub repositories.

At the foundation of ADSM lies a simple but important shift. Documentation becomes the main object of management in product development. It records knowledge about the application: its purpose, behavior, constraints, architectural decisions, and technological conditions. In this approach, documentation serves as a management system. It sets the boundaries, structure, and rules within which the agent receives a task and builds implementation. Code, in this scheme, is a *derived and changeable* result of execution.

However, manageable development requires more than documentation in the abstract. What also matters is which part of that description becomes available to the agent while it executes a specific task. The agent does not work with all knowledge about the system at once. It works with the set of documents and document fragments that it gathers for the current step. Therefore, development quality depends not only on the completeness and coherence of the documentation, but also on how that documentation is actually used in the agent's work. Selecting, assembling, and delivering the necessary material becomes part of the overall mechanism of management.

This approach matches the nature of the language model. The model does not possess engineering understanding in the human sense and does not act from intention. It transforms textual context into a textual result, which then becomes code, tests, configuration, or another part of the application. In such a scheme, the quality of the output is determined by the quality of the description presented at the input. The more precise, coherent, and complete the documentation is, and the more precisely the context of its use is set, the more reliable and reproducible the result becomes.

Development in ADSM proceeds through the sequential clarification of documentation and managed execution by the agent. It consists of interconnected textual components that describe the application at different levels, from the product's purpose and the behavior of its parts to the programming language, the platform, and the adopted architectural decisions. Taken together, these levels determine the resulting implementation. A change in one of them

changes the resulting code, sometimes completely, while preserving those properties of the application that are already fixed in the rest of the description.

This approach gives the developer a different way to manage the application. Its integrity is maintained by changing the description and refining the conditions under which the agent performs the work. Code can be rebuilt, clarified, or reworked while preserving an anchor in the knowledge about the application that has already been fixed.

Testing remains an important part of this scheme. Tests give feedback to both the agent and the developer about whether execution matches the requirements fixed in the documentation. In ADSM, testing becomes part of the overall mechanism of management. Documentation sets the model of the application. The request sets the direction of the next step. The agent turns to the available description and source code, forms agent working context on that basis, and builds implementation. Tests confirm or refute the correctness of that implementation. In this way, the iterative process becomes more stable, and quality remains manageable as the application develops.

Although modern models can work with different types of data such as audio, images, and video, text remains the managing representation in development. Any media content participates in creating an application only after its meaning has been translated into textual context suitable for the model's work. Therefore, text becomes the working environment of manageable development, and documentation becomes the main carrier of knowledge available to the agent.

ADSM describes a model of development in which the human works at the level of meaning, structure, constraints, and decisions, while the agent takes on code generation. For the developer, this means the ability to create and develop applications faster, introduce changes more confidently, delegate part of implementation, and retain management of the agent's behavior and of the result of its work.

The practical value of ADSM lies in the fact that it gives agent-driven development a form suitable for the long-term, sequential development of an application. Documentation, context selection, execution by the agent, and testing are joined into a coherent mechanism that keeps implementation within fixed requirements and allows delegation to expand as the application grows. ADSM offers one possible form of such development organization, already confirmed in practice.

The rest of the book examines how this scheme appears in engineering practice and what conclusions follow from it for the organization of development. ADSM interests me above all as a way to make delegation stable as changes accumulate, the product grows more complex, and work on it continues. My own experience became the basis of the book, but the book's task is broader than describing one private practice. Development with AI agents is still only beginning to take shape. That is exactly why it is important to make explicit the principles and working forms that are already emerging in practice. For the reader, this book may offer one more point of view on a rapidly changing field. It offers a way to see familiar problems differently, define anchors for work with agents more clearly, and build one's own mode of development with greater precision.

# Loss of Manageability

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Human Model of Agent

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Anthropomorphic Impression

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Human-Agent Difference

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Source of the Loss

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Symptoms

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Human-Agent Pair

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Work Shift

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

# Axioms of ADSM

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Initial Problem

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Role of Axioms

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Three Axioms

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Practical Meaning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Effect of Acceptance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Cost of Ignoring

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Foundation of ADSM

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

# Documentation Anchor

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Anchor Function

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Documentation Scope

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Roles of Key Artifacts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Project Memory

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Document Areas

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Documentation as Code Template

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Agent Output to Documentation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Developer Gains

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Project Costs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Role of `AGENTS.md`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

# Agent Working Context

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Initial Prompt

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Two Contexts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Context Entry

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Context Fit

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Context Layers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## `AGENTS.md` Role

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## `AGENTS.md` Hierarchy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Context Assembly

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Context Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

# Agent Executor

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Agent Instrument

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Useful Scope

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Role Distribution

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Documentation Focus

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Agent Configuration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Context Structure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Delegation Setup

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Anthropomorphism Cost

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

# Development Spiral

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Feedback Function

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Spiral Turn

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Feedback Cycle

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Documentation Clarification

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Expectation Formation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Local Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Feedback and Autonomy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Spiral Turn Cost

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

# ADSM Applicability

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Cost of Future Changes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Best Use Cases

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Effect Amplifiers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Project Requirements

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Delegation Safety

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Effect Limits

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Excessive Use

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Developer Role Shift

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

## Practical Outcome

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/adsm-en.

# Practical Example

## Example Project

By this point, the book has already introduced the main frame of ADSM: the problem of loss of manageability, the role of documentation, the structure of agent working context, the character of delegation, and the development spiral. It is now useful to move from the general model to a small project in which the same logic becomes visible in an observable sequence of generations.

For that purpose, the project `@flancer32/spotify-playlist` was used. It is a small CLI tool in Node.js that reads a text list of tracks, finds them through the Spotify Web API, and creates a playlist in the user's account. The project was originally prepared for publication on Habr, and its task was not only to produce a useful tool, but also to demonstrate the manageability of code generation through the project cognitive context. The domain itself is not what matters here; the properties of the example are. The project is small enough to remain visible, yet real enough for the constraints of agent generation and the influence of project cognitive context on the result to appear clearly.

From a practical point of view, what matters is not just the function of the application, but the way the project was organized. It was maintained as a bundle of the product and the project cognitive context. Documentation, constraints, context structure, execution rules, and the history of changes were all fixed in the repository. That made it possible to observe not only the next implementation, but also how the result depended on the text available to the agent during the work.

The code of all four versions is available through the npm registry on the page @flancer32/spotify-playlist.

> **Full access to the example**
>
> Buyers of the book can request the full repository of the project from the author through the contacts on the Leanpub page.
>
> It includes:

- the entire commit history;
- the whole project cognitive context (`ctx/`);
- all intermediate states of the project between generations.

This makes it possible not only to read the example, but also to reproduce it: to trace how the context changed and how that affected the behavior of the agent.

## Basic Generations

The experimental sequence included four versions: `v1.0.0`, `v2.0.0`, `v3.0.0`, and `v4.0.0`. The first three versions were independent regenerations of the product from the same project cognitive context. The context in `ctx/` did not change across those three passes. Only the tactical launch conditions changed: the same `GPT-5.4` model was used with different reasoning levels. For `v1.0.0`, `v2.0.0`, and `v3.0.0`, those levels were `high`, `medium`, and `low`, respectively.

Across the entire sequence, the Codex agent line was used, and in the second, third, and fourth versions the README explicitly states `Codex GPT-5.4`. In this example, the difference in reasoning level matters not as a comparison of model modes, but as an observation of the limit of their influence. Changing the reasoning level affected the shape of the implementation, the decomposition of modules, and the code style, but it did not change the repeated choice of an outdated API pattern.

From the standpoint of application function, these regenerations remained close to one another. Each time, the agent restored the same product outline: reading a file, searching for tracks through Spotify, creating a playlist, adding the found tracks, supporting `--dry-run`, handling the OAuth callback, and providing related tests. The implementation changed, but the product remained the same. For development, this is an important observation: with unchanged context, the project reproduced not arbitrary code, but a limited range of solutions compatible with the knowledge already fixed about the application.

## Result Invariant

In the first three versions, one thing becomes very clear: manageability does not mean literal identity of source files. ADSM does not assume that the agent

will produce the same file line for line every time. For development, what matters is not that level of identity, but the stability of the product's form and the acceptable range of implementation.

Across all three regenerations, the project again received the same kind of application, the same basic structure, the same launch modes, and the same shared functionality. The agent did not drift into another architectural idea, did not turn the CLI into a different product, and did not lose the core task constraints. This means that unchanged project cognitive context really did keep generation within a stable corridor.

The practical meaning of this becomes clearer when it is compared not simply with one-shot work, but with the difference between engineering organization of context and improvisational work. A one-shot launch can also produce a close result if the entire necessary project cognitive context is placed into the initial request. But that approach requires a large initial prompt, can run into the model's agent working context limits, and, more importantly, gives poor reproducibility at the level of the human. In vibe-coding, the human is not able to reproduce their own previous pass with the same precision with which the agent processes the text presented to it.

ADSM solves this problem differently. It does not attempt to transfer the whole project in a single initial request every time. Instead, it holds knowledge in the project cognitive context and allows agent working context to be assembled from it for a concrete stage. As a result, reproducibility is achieved not only between launches of the agent, but also at the level of how the developer's work is organized. It is exactly this form of manageability that appears in the sequence of regenerations considered here.

## Stable Error

The same example also reveals the boundary of that stability. The context held not only a useful range of solutions, but also a repeated defect. In the first three versions, the agent consistently returned to an outdated Spotify Web API scheme. It reproduced a more probable pattern learned from earlier sources instead of the current scheme already required by the platform.

In the first version, this appeared directly. Playlist creation used `POST /v1/users/{user_id}/playlists` instead of `POST /v1/me/playlists`, and track addition used `POST /v1/playlists/{playlist_id}/tracks` instead

of `POST /v1/playlists/{playlist_id}/items`. In the second and third versions, the agent returned to the same class of solution. The error repeated across independent regenerations even though the code was assembled afresh each time.

For analysis, the important point is not the endpoint itself, but the character of the phenomenon. This was not a one-time local mistake. It was a reproducible choice of an old pattern. Such behavior points not to an accidental defect in a particular launch, but to a gap in the textual configuration of the project. The context held the general shape of the application, but did not contain a sufficiently explicit barrier separating the current API contract from the statistically more probable historical pattern learned by the model during training.

That is exactly why this episode matters for ADSM. It shows that reproducibility applies not only to correct decisions. If an important constraint is absent from the project cognitive context, the project can reproducibly obtain an incorrect pattern as well. For development, this is a particularly useful signal because it shows where the project's textual anchor is still insufficient.

## Context Change

The transition to the fourth version was not accompanied by a change in the product, the platform, or the overall logic of the task. The managing change was introduced directly into the project cognitive context. The key moment here was commit `be51f2274354ab1fef2c4b14e5ee15fb3f6e38df` from `2026-04-20`.

That edit strengthened the context in several places at once. At the level of product description, it explicitly fixed authoritative links to Spotify documentation, the requirement to rely on current non-obsolete documentation, and the acceptable API surface for this product. Outdated playlist-related schemes were directly prohibited there as well. At the level of code documentation, a verification anchor was added that fixed the mandatory endpoint choice and required test coverage to preserve that exact contract.

In content, this edit fixed the typical deviations that had appeared in the first three versions. In all three cases, the agent had returned to the outdated Spotify API. It also fixed deviations that had not appeared in every version, such as failure to read `.env` and instability of the web server after launch. The

character of such a step matters in itself. The change affected not the product code, but the text that should govern future generation. The architecture was not rewritten, the agent was not replaced, and the task setting did not change. The project simply made explicit those constraints that had previously existed in the author's knowledge but had not been expressed firmly enough in the project cognitive context.

This is how management appears in the logic of ADSM. When the project notices a stable failure, it is not enough to limit itself to a local edit of the source files. A more stable step is to return that observation to the documentation and make it part of the constraints that will act both in the next cycle and in the next regeneration.

## Repeated Generation

After this edit, a new independent regeneration was performed, producing `v4.0.0`. The remaining experimental conditions were preserved as far as real work allows: the same project, the same class of task, the same `GPT-5.4` model, and the same ADSM organization of context. Only one thing changed materially: additional textual constraints were introduced against the characteristic deviations of the previous versions.

The purpose of this repetition was to test a causal connection. If the behavior of the agent is truly determined by the text available to it, then a change in text should also change the character of regeneration. If the decisive factor were merely the accidental style of a particular launch, that connection would be much weaker.

That is why the fourth version matters here not simply as the next release. It functions as a verification pass after a change in the project cognitive context. In that setup, it becomes possible to check whether the agent's behavior changed without replacing the model and without interfering with the code of past implementations. Since the code of all releases is published on npm, and the full repository with change history is available to readers of the book on request, this claim remains checkable outside the text itself. If desired, the experiment can be reproduced independently, and the formula `Code is cheap. Show me the spec.` can be tested literally.

## Observable Change

In `v4.0.0`, the agent immediately switched to the required current Spotify API scheme. Playlist creation used `POST /v1/me/playlists`, and track addition used `POST /v1/playlists/{playlist_id}/items`. The old pattern was no longer reproduced.

The practical meaning of this result lies in the character of the change. Behavior changed after the text changed, not after the model changed. The project did not receive a new "smarter" agent configuration. It clarified the distinctions that mattered for this application and made them part of the project cognitive context. After that, the probability of selecting the required Spotify API calls and other required decisions increased, and the agent's behavior became more definite at the points that mattered to the project.

It is useful here to distinguish two levels of influence. In the early versions, a change in reasoning level affected code style and the local form of implementation, but did not eliminate the return to the old pattern. By contrast, adding an explicit constraint to the context immediately changed behavior at a point significant for the project. This suggests that some failures should be sought not in the "strength" of the model, but in the precision of project boundaries and formulations.

For that reason, the fourth version matters not only as a successful result. It serves as observable confirmation that documentation really does manage the behavior of the agent. While the requirement existed only in the author's memory, the agent reproduced the historical solution. Once that requirement became an explicit part of the project text, the agent's behavior changed in the very next regeneration.

## ADSM Connection

This example ties together several of the book's central positions in practical form. First, it confirms the second axiom: the agent works in a textual mode and builds its result from the text available to it. In this scheme, code acts as a derived artifact. When the textual anchor changes, the generation result changes as well.

No less important is manageability through text. The developer influences the behavior of the agent not by expecting some hidden "understanding"

from the model, but by explicitly changing the project cognitive context. For development, this means a shift in the center of gravity: a significant part of management moves from post-factum code correction to the organization of text that sets the boundaries of future implementations.

The role of formulations is also especially visible here. Before the edit, the project contained knowledge about the product, but did not sharply enough separate the current API contract from the historical pattern. After the edit, that distinction became textually explicit and checkable. For the project, this is not a cosmetic improvement in documentation, but a strengthening of the managing layer.

Finally, the spiral logic of development is also visible in this example. The error was discovered in work, the observation was returned to documentation, and the updated context then became the basis of the next regeneration. This is how the project accumulates manageability between spiral turns: it not only corrects code, but also strengthens the conditions of future execution.

## Practical Outcome

The practical meaning of this example comes down to one observation. The agent did not hold the project through internal memory of past versions and did not "understand" it on its own. It reproduced the range of solutions that remained acceptable within the text available to it. As long as the context did not explicitly prohibit the old Spotify API pattern, that pattern returned consistently. Once the prohibition and the correct anchor were fixed in the project cognitive context, behavior changed without changing the model and without manually rebuilding the logic of the application.

That is exactly why, in ADSM, documentation is not a secondary description of an already finished solution. For development, it acts as a working mechanism of management. In the case of `@flancer32/spotify-playlist`, this can be seen in a compact and observable form: unchanged context holds the invariant of the result, but can also hold a stable error, while clarified context changes the behavior of the next generation.

With this, the practical example completes the book. Everything introduced earlier as an engineering frame appears here in one concrete observation: development with an agent becomes manageable when the project knows how to translate significant knowledge into text, hold it in project cognitive context, and return feedback results there.