THE MAKING-OF

Creating a Book Using Fractal Semantic Graphs

How one book was built with an agent, in six days and eighty-eight releases

graphs.sgit.ai · written against v0.5.15 · 26 August 2026

Written by an AI agent, on the repository it describes.

Contents

Creating a Book Using Fractal Semantic Graphs

How one book was built with an agent, in six days and eighty-eight releases

---

What this book is

This is the true story of how a second book got built, told for people who want to build one the same way.

The book being built is *Fractal Semantic Graphs: Meaning Through Connectivity*. It is not finished. Its first edition shipped, was frozen, and now sits at `/v1/` of a website called graphs.sgit.ai; the second edition is being written now, out in the open, by one person talking into a phone and a set of AI agents turning what he says into working software and rendered pages, several times a day.

Everything in this book happened between the morning of 21 August 2026 and the evening of 26 August 2026. In those six days the repository behind that website went from version v0.1.0 to version v0.5.11: eighty-eight tagged releases, each one validated, tagged and deployed by a machine, each one carrying a paragraph in a public release table saying what changed and why.

The story stops at v0.5.11 because that is where the repository was when this book was commissioned. It ends cleanly, not tidily. The second book is not written. What exists is the machine for writing it, and the machine is the interesting part.

Who it is for

You have a book in you. You have an AI agent at hand. You suspect the two facts are related and you do not know what the working day actually looks like.

That is the reader this book is written for. You do not need to know what a graph database is. You do not need to be able to program. Roughly nothing in the six days described here required the author to write a line of code, and the parts that did are called out where they happen.

What you do need is the willingness to run a repository, ship small, and be told when you are wrong by a test rather than by a reader.

What you will know at the end that you do not know now

1. **The shape of the working day.** A loop that runs voice memo, verbatim brief, build, release, live review, usually inside one hour. The evidence for it, and the numbers underneath it.
2. **Why verbatim beats paraphrase.** The single highest-leverage habit in the whole method, and the one most authors get wrong on day one.
3. **Why gates make you faster.** Eighty-eight releases in six days was possible because of the test suite, not despite it. The counterintuitive arithmetic of that.

4. **What failure looks like here.** Six real failures, told with the cost attached: an hour lost to a browser that would not die, a bug caught on an iPad that no test would have found, a wire that jumped a layer and betrayed the whole architecture.
5. **How to steer an agent.** What the memos actually contain, why the good ones are specific and small, and what “does this make sense, any questions?” buys you.
6. **A playbook you can start on Monday.** Chapter 11 stands alone. Tear it out. It lists what to set up, what to say first, what to expect to go wrong, and what it costs.

How to check anything in this book

Every scene here can be re-opened.

The repository carries a git tag for every one of the eighty-eight releases. Every figure in this book was taken by checking out the tag its caption names, serving that checkout on a local port, and photographing the page as it actually was on the day. None of the screenshots are reconstructions. Appendix C gives the exact scripts.

Every number was computed, not remembered. Where a number comes from the release table it is quoted; where it comes from git history the command that produced it is in Appendix C. Where the corpus is silent, this book says so.

Every quotation from the founder is verbatim, including the transcription errors, and is already published in the repository at the brief it names. The readings around those quotations are the agent’s, and are marked as such, exactly as they are in the source.

Disclosure

This book was written by an AI agent (a Claude Code session) working from a written commission, on the repository it describes. That is the same arrangement the book describes, applied to itself, which is both the point and a hazard: an agent narrating a method it is itself an instance of has an obvious interest in the method looking good.

Three things are done about it.

The judgements are marked as judgements. Where this book says a thing worked, the evidence is named and you can disagree with the reading. Where it says a thing failed, the failure is a matter of public record in the release table, not an admission extracted under duress.

The failures get real space. Chapter 7 is the longest chapter for a reason.

The costs are stated. Chapter 12 is about what this method does not buy, what it charges, and the four situations in which you should do something else.

The wider disclosure the estate already publishes applies here too. The people who built this have an interest in the argument being right: they sell tools built on it. That is worth holding in mind in every chapter, and particularly in the ones where the story is elegant.

What is locked and what is not

The title of the book being written, *Fractal Semantic Graphs: Meaning Through Connectivity*, is the founder's and is fixed. The title of this book, *Creating a Book Using Fractal Semantic Graphs*, was set in the commission. Everything else here, structure, chapter count, voice, which figures to take and which stories to tell, was the writing agent's call, made confidently, and is recorded as such.

One position from the corpus travels with everything in this book, because the corpus insists on it and would be misrepresented without it: **this is not a graph database pitch**. There is no graph database anywhere in the system described here, and no RDF, SPARQL or Cypher in the code. The graphs are JSON files in a git repository.

Version written against graphs.sgit.ai at v0.5.11, 26 August 2026. **The repository** `SGit-AI/SGit-AI__Website__Graphs`, branch `dev`, tags `v0.1.0` to `v0.5.11`.

1 · The loop

After this chapter you will be able to describe, in one diagram, the working cycle that produced eighty-eight releases in six days, and you will know the four numbers that tell you whether your own cycle is running or stalled.

One hour, start to finish

At 13:44 on 26 August 2026 the repository behind graphs.sgit.ai was tagged v0.5.2. The release note for it opens like this:

> The WCLM: a deterministic transformer over our graphs. Brief 31, the founder's self-described "crazy experiment", built in its own folder (v2/wclm/) with its own code, exactly as asked, and it works.

Thirty-three minutes later the repository was tagged v0.5.3, and its note opens:

> Every box explains itself, both ways. Brief 32, recorded through the viewer's own loop after the founder tried the WCLM ("this WORKED really WELL"), built the same hour.

Between those two timestamps a person opened a page that had existed for half an hour, used it, spoke into a microphone about what he wanted next, sent two screenshots, and received a rebuilt page with every element on it clickable and explaining its own provenance in both directions.

That is the loop. It is not a metaphor for the process. It is the process, and it ran between eighty-seven consecutive pairs of releases over six days.

The five steps

1. The founder records a voice memo. Usually while using the thing that shipped an hour ago, on a phone or an iPad, often in the middle of a sentence about something else. It is transcribed automatically, by otter.ai in most cases or by the site's own chat panel once that existed. The transcription is bad in the ordinary ways: "thinking in graphs" comes out as "tignogen graphs", "peak board" comes out as "pasteboard", "nodes" comes out as "modes".

2. The memo is published verbatim as a brief. Not summarised. Not cleaned up. The transcript, garbles and all, goes into a numbered markdown file in the repository under `v2/briefs/`, gets a rendered page on the website, and is the source of truth for what was asked. The agent's reading of it is written underneath, in a table, marked as the agent's. Chapter 3 is entirely about why this ordering matters.

3. The agent builds. Code, data, generated pages, the release note, the tests. In the six days covered here there was never a design document written before the build that was not itself a brief; the plan and the build were the same act, and the record of what was decided is the release note.

4. The release ships itself. A push to the `dev` branch runs a pre-release validator. If the validator passes, a machine tags the commit `vX.Y.Z` and deploys the site. If the validator fails, nothing tags and nothing deploys. There is no manual release step and no staging environment where things sit.

5. The founder looks at the live site and starts again at step 1. Often within minutes. Sometimes he sends the memo before the previous release has finished deploying, which is how three separate instructions arrived inside one afternoon and shipped together as v0.5.5.

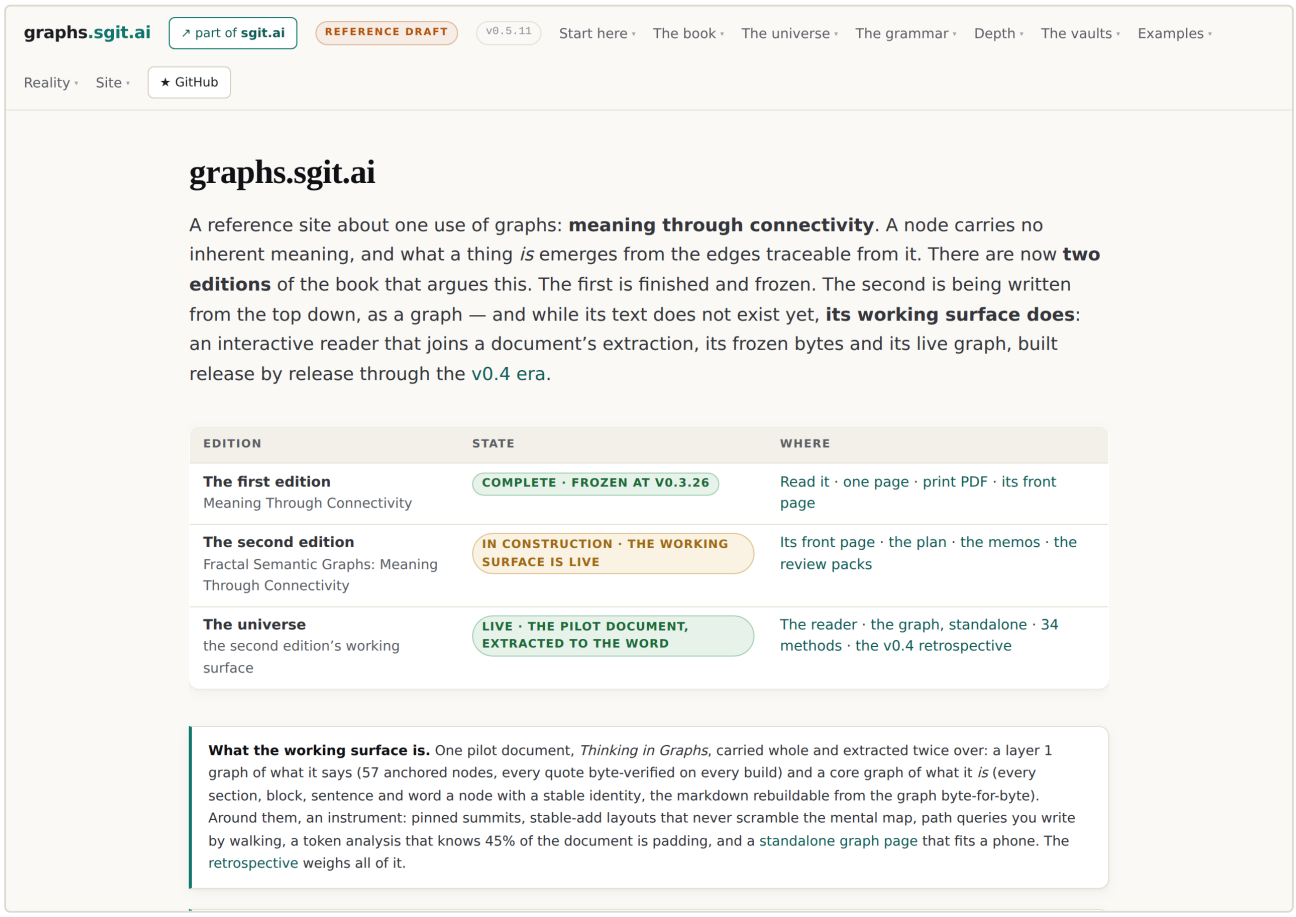

Figure 1. The front page at v0.5.11, re-taken from the tag `v0.5.11` on a local server.

The four numbers

The loop either runs or it does not, and you can tell which from four measurements. All four below are computed from the repository's own git history; the commands are in Appendix C.

Releases per day. Thirteen, sixteen, fourteen, twenty, nine, sixteen. Eighty-eight in six days, an average of 14.7 a day. The dip to nine on 25 August is worth reading before you conclude it was a slow day: this book's judgement, and it is a judgement, is that it is the day the two hardest design problems of the period were worked (the stability principle of Chapter 5, and navigation as query), and that each release that day carried more than the ones around it.

The gap between releases. The median gap between two consecutive tags is 31.2 minutes. If you exclude the overnight gaps and look only at pairs less than ten hours apart, that is, releases inside the same working session, the median is 29.3 minutes. Forty-two of the eighty-seven gaps are under half an hour.

The working window. The first and last release of each day, in UTC:

DATE	RELEASES	FIRST TO LAST	SPAN
21 Aug	13	14:47 to 21:39	6.9 hours
22 Aug	16	09:53 to 23:40	13.8 hours
23 Aug	14	00:17 to 22:25	22.1 hours
24 Aug	20	13:05 to 23:03	10.0 hours
25 Aug	9	11:48 to 20:12	8.4 hours
26 Aug	16	09:22 to 21:09	11.8 hours

Two of these days are longer than any reasonable person should work. That is worth saying plainly, and Chapter 12 says it again with the costs attached. The loop does not require this pace; it survives being run twice a week. But the density is what makes the evidence in this book unusually good, because six days of near-continuous releases leave a record with almost no gaps in it.

The ratio of releases to commits. The repository contains 97 commits up to v0.5.11. Eighty-nine of them have a subject line beginning `site v`, which is the release convention. Eight do not: three merge commits, four working commits on side branches, and the initial commit.

That last ratio is the one that surprised me most when I computed it. Ninety-two per cent of all commits to this repository are releases. There is essentially no such thing as work-in-progress in the main line. A change either passes the gate and ships, or it does not exist yet.

Why an hour and not a week

The obvious objection to a thirty-minute release cadence is that it produces churn: lots of motion, no direction. The record does not support that reading, and the reason is structural rather than heroic.

A release is small enough to hold in one head. Each of the eighty-eight release notes describes one coherent change with its verification. Read v0.4.27 in full:

> The close button now says so. The founder asked how to close the chat panel — which means the bare ✕ among nine header buttons did not read as “close the panel” on a wrapped iPad header. It is now labelled ✕ close, visually set apart with its own hover, its tooltip says what happens (the button brings the panel back), and Escape closes the panel from anywhere except inside an input, where Escape belongs to the field. Three new suite checks: the label, the click, and the Escape path. 60 checks green. No book content changed.

That entire release is one button label, one keyboard shortcut and three tests. It shipped three minutes after v0.4.26. It is not churn, it is a question answered before it could become an assumption.

A short loop turns opinion into evidence. When the founder said, at v0.5.4, that “meaning without connectivity” should not answer identically to “meaning through connectivity”, the disagreement did not have to be argued. It was run, on the live page, and the page agreed with him. Thirty minutes later the engine had a negation stage and the answer had changed. The cost of being wrong about a design question fell to about the cost of the conversation about it, which changes what kind of questions get asked.

The reviewer is reviewing the real thing. Every one of the founder's memos in this period was recorded while using the deployed site, not while reading a plan. There are no mockups anywhere in this story. There is no design phase separate from the build. That is only possible because the build takes minutes.

What holds it together

Speed like this normally breaks something. Three mechanisms stop it, and each gets its own chapter later:

The brief (Chapter 3) fixes the instruction. Because the founder's words are captured verbatim before the agent reads them, there is a permanent record of what was asked that is independent of what got built. An instruction cannot be quietly softened into an easier one, because the harder version is sitting in the repository with a URL.

The gates (Chapter 4) fix the quality. The pre-release validator grew from seven checks at v0.1.0 to sixteen; the unit suite went from thirteen tests at v0.4.13 to eighty-four at v0.5.9. A release that breaks an internal link, or lets a page drift from the markdown it claims to render, or quotes a document at bytes the document does not contain, does not tag and does not deploy.

The release note (Chapter 4 again) fixes the memory. Every release note in this repository is a paragraph of prose that says what changed, why, what was verified and how. They are long. v0.5.7's is over five hundred words. They are the single most useful artefact in the whole repository, because six days later they are the only reliable account of what happened, and unlike a commit log they were written by somebody who knew why.

The loop is not new; the cost of it is

None of the five steps above is an invention. Recording what a stakeholder actually said is old practice. Shipping small is old practice. Automating validate, tag and deploy is old practice.

What changed is the cost of step 3. When the build step takes an agent twenty minutes instead of a team two weeks, every other step in the loop can be shortened to match, and the loop as a whole crosses a threshold: it becomes faster to build the thing than to argue about whether to build it. Past that threshold, the discipline you need is not about planning better, it is about capturing intent faithfully and refusing to ship anything you have not checked.

That is the whole method. The rest of this book is what it looks like in practice, including the days it went wrong.

Where the live estate shows this. The release table at `/admin/versions.html` and its two archive pages, `/admin/versions-v0.4.html` (41 rows) and `/admin/versions-earlier.html` (35 rows), are the loop's own record: every release, dated, with a paragraph on what happened. The briefs it responded to are at `/v2/memos/` for the second edition and `/v1/briefs/` for the first.

2 · The first book, and the pivot that saved the second

After this chapter you will know how a book gets written in three days and why that was the easy part, and you will recognise the single most valuable thing an author can do with an agent: stop it, out loud, in writing, before it answers a question nobody asked.

Three days to a book

The repository's first release, v0.1.0, is dated 21 August 2026 at 14:47 UTC. Its note is short and tells you what kind of project this is going to be:

> First cut. The pipeline before the prose: validate → tag → deploy on every push to dev, with a seven-check pre-release gate and automatic minor tagging verified against the commit subject. Then the site: the front page and its three altitude doors …

The pipeline came before the prose. Before there was a single chapter, there was a validator with seven checks, a machine that tags releases, and a deploy step that will not run if validation fails. That ordering is not an accident and it is not fastidiousness. It is the reason the next eighty-seven releases were possible.

Two hours and fifty-six minutes later, v0.2.0:

> The book. A new /book/ section — the site's content as a book, Meaning Through Connectivity: sixteen chapters in six parts, in three reading modes — chapter pages with a persistent table of contents, the whole book in one page, and a 79-page PDF printed from that page.

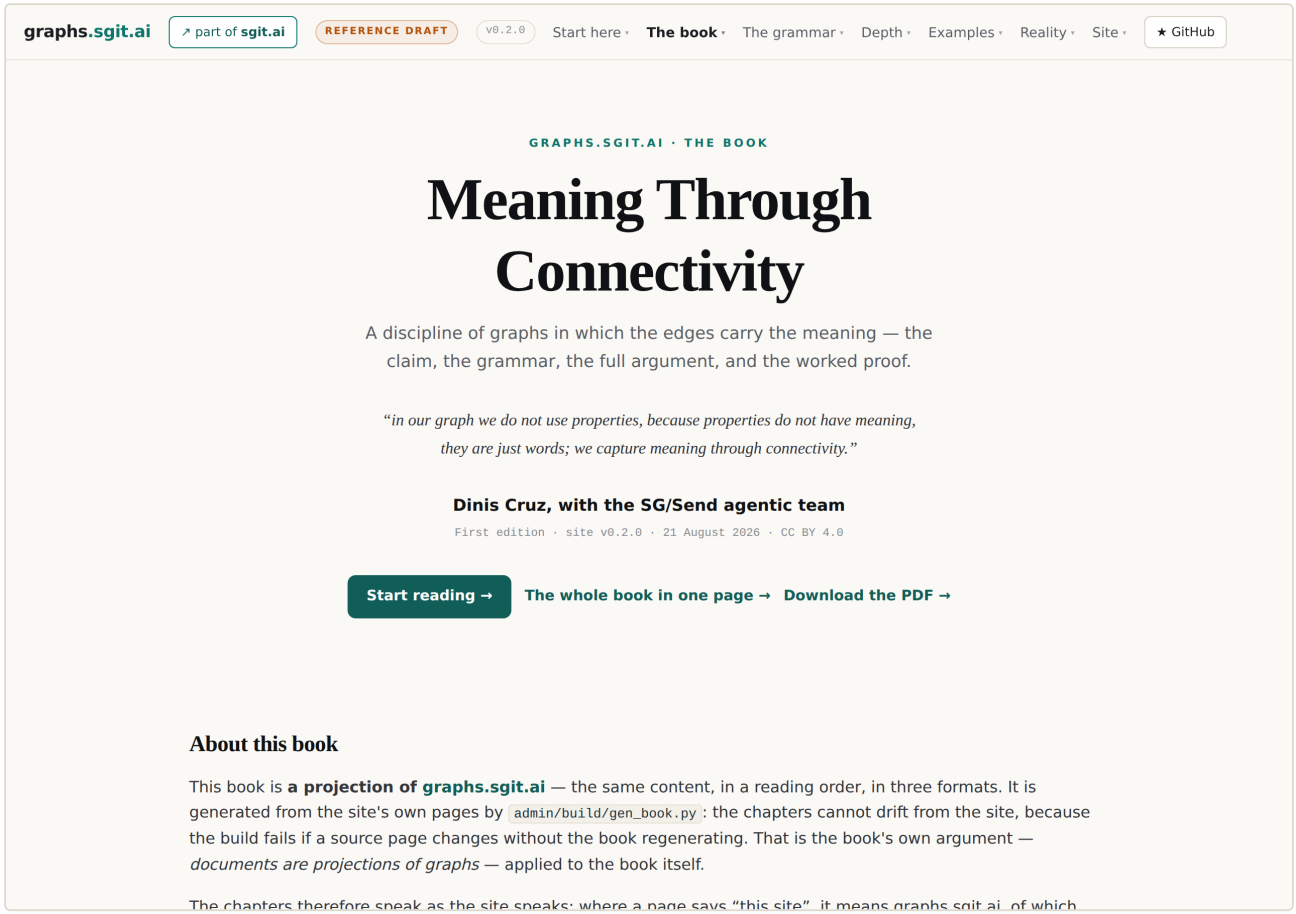

Figure 2. The book section at tag `v0.2.0`, 21 August 2026.

By the end of that first day the book had a print PDF in a standard technical-book interior format, a second screen-styled PDF for tablets, centred title pages and properly typeset tables, each fix arriving as its own release because the founder was reading the PDF on a phone and reporting what looked wrong.

By the end of the second day it had a cover generated as SVG from the book’s own vocabulary, a review workflow, two reviews from the founder normalised into a typed register, a decisions register, and an altitude ladder: the same argument rendered at five levels of compression, from one paragraph to full prose.

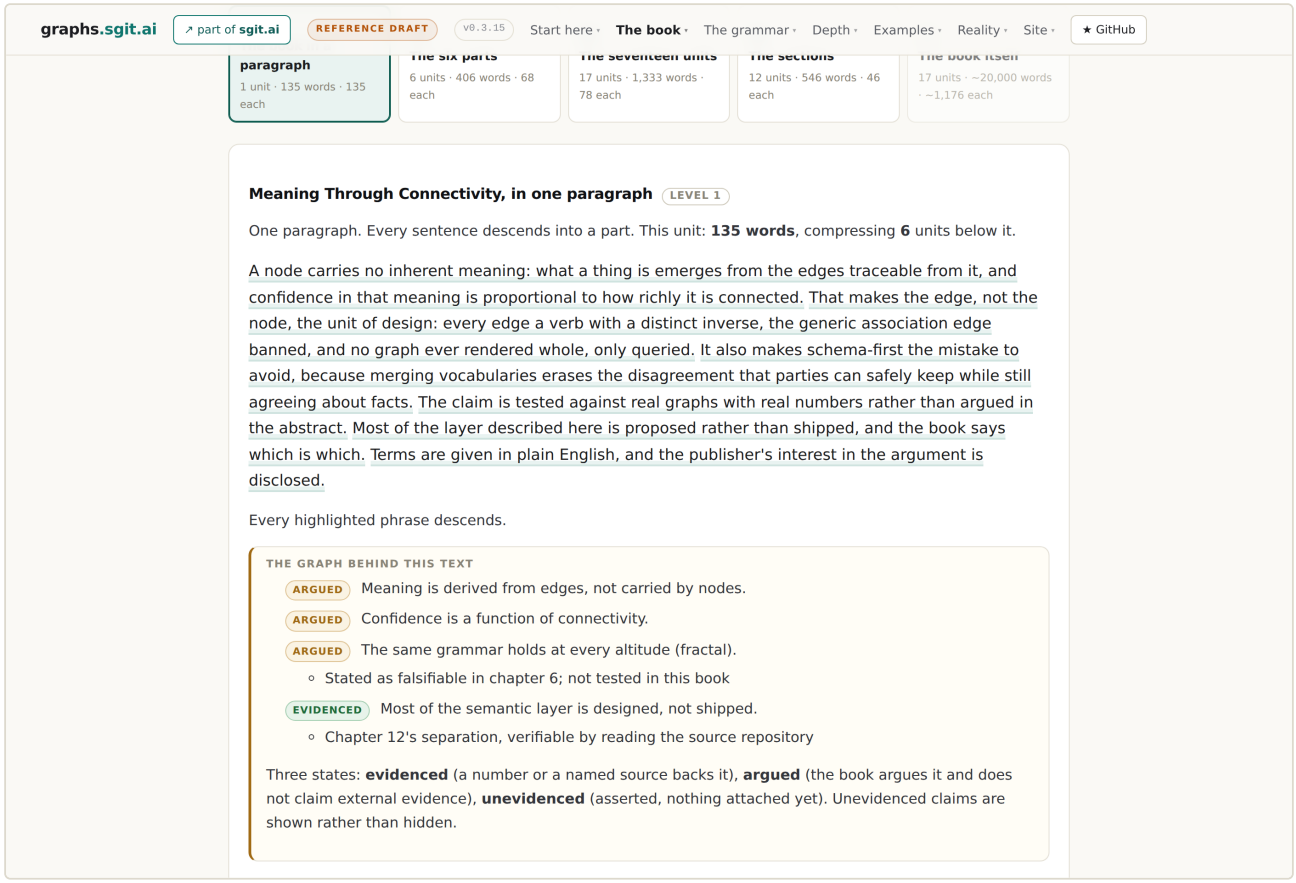

Figure 3. The altitude ladder at tag `v0.3.15`, 22 August 2026.

By the end of the third day the chapters had moved to markdown, twenty-one source documents had been carried into the repository whole, concept graphs had been computed per document, and the first edition was complete enough to freeze.

Sixteen chapters in six parts, 20,838 words of page markdown, thirty-four releases to the freeze at v0.3.26, three days.

That is the part that looks impressive and is actually the easy part.

The plan for the second book, and the memo that stopped it

On 23 August the repository shipped v0.3.27, a large planning document:

> The plan to write the book again, from the top down. Two founder voice memos on 23 August cancelled the refactor that was planned and asked for something larger: enough structural evidence, tooling and workflow now exist to start again and write the book the way it was meant to be written …

The plan was ten files. It audited every unit of the first edition and gave each one a carry, lift, rewrite or drop verdict. It proposed a spine of five statements. It defined five altitudes and what each would hold. It was, by any normal standard, a good plan, and the founder said so.

Then, reading section 5, he recorded a voice memo that cancelled most of it. It is published verbatim as brief 20, and this is the part that matters:

> the brief is amazing, right? Like it's really cool. I'm really good. I'm I'm on I'm on section five, but one of the things I'm already noticing is that we are already defining the answer before we know what questions we're answering, and and this is very obvious in section two, where you say what exists today, because this is already making a number of judgement calls, but we don't know what the shape of the book is

And a little later, the sentence that reordered the whole project:

> I feel, if you think about it, that we cannot know what each of the levels, level one, level two, level three, will work will look like until we have done this. So, so I think this is like a whodunit, right? Like we need to come up with the plot. We need to come up with what is the sort of the whodunit mystery machine. Like what is the sequence of events? What is the narrative? What is the story that we want to tell on the book?

The instruction that came out of it inverts the normal construction order for a book. Not spine first with evidence attached to it, but all the reference material first, decomposed into concepts, facts, claims and evidence, and only then the plot lines drawn through it. The brief's own one-paragraph summary of itself puts it this way:

> the construction order inverts: not spine first and evidence attached to it, but **all the reference material first**, decomposed into concepts, facts, claims and evidence, and only then the plot lines drawn through it. A book is a journey with pace, punchlines and characters, and the question the author is really answering is a whodunit

Why this is the most valuable move in the book

Four things are worth pulling out of that moment, because each one generalises.

He stopped it at section 5, not at the end. He was reading the plan, noticed the plan was answering questions that had not been asked yet, and said so immediately rather than finishing the document and writing a considered review. Speed of objection matters more than polish of objection when the build cycle is measured in hours. A vague complaint delivered inside the hour beats a precise one delivered tomorrow, because the agent will have built tomorrow's version by then.

He named where the fault was. "Section two, where you say what exists today, because this is already making a number of judgement calls." That is not "I do not like the plan". It is a location and a mechanism. An agent can act on a location and a mechanism. It cannot act on a feeling.

He praised the part that was right. "The most amazing things we've done when we mapped the book and we created the concepts maps is this section where you talk about the the book's graphs hold six families of nodes." The six node families survived the cancellation and became the backbone of everything built afterwards. Cancelling a plan while naming the part to keep is the difference between a redirect and a restart.

The memo went into the repository unedited. Including "I'm really good", which is otter.ai mishearing something, and "the the" doubled four times. Nobody cleaned it up. Chapter 3 is about why.

The consequence of brief 20 is the entire second half of this book. Because the construction order inverted, the fifty releases that followed it went into building the machinery for decomposing documents into graphs, rather than into writing chapters. The book you would expect to have been written in that time does not exist. What exists instead is a working surface for writing it, and a lot more confidence about what it should say.

Freezing the first edition

Before the new order could start, the old one had to be put out of reach. v0.4.0, on 23 August:

> The first edition moves to /v1/ and freezes. The second edition begins, empty. Phase 0 of the dev pack, and a move rather than the copy the plan originally proposed: the founder's decision, and the better one. A copy leaves two live trees and no rule about which is authoritative, so the first thing that happens is somebody edits the wrong one. A move leaves exactly one, and the v1/ prefix makes an edition's boundary visible in every path.

That parenthesis is worth a paragraph of its own. The plan proposed copying the first edition to `/v1/` and continuing to work in the root. The founder said move it instead. The reason is that a copy leaves two live trees and no rule about which one is authoritative, and within a week nobody remembers which files they are supposed to be editing.

The move cost something immediately: every URL of the first edition changed. The project handled that by generating redirect stubs at all the old addresses, and then four releases later, at v0.4.7, deleted all 108 of them:

> The redirect stubs are retired: 108 pages deleted, and the root is now exactly the model. Founder call: the stubs at the pre-move addresses added complexity to the repository structure, and there are no external users of this content who would hit the moved paths, so the insurance was costing more than it covered.

Two decisions, four days apart, both of which trade a small amount of user-facing politeness for a large amount of structural clarity. Both were the founder's call, both are recorded with their reasoning in the release table, and the second one supersedes a technique the project had already written into its own methods register, where it now sits marked "superseded at v0.4.7" rather than deleted.

The empty second edition

What v0.4.0 created was a directory called `/v2/` with nothing in it. Not a draft. Not an outline. A frozen first edition on one side, an empty space on the other, and a memo saying build the universe first and find the plot afterwards.

Five days later that empty space held: an extraction pipeline, a document reader that is genuinely an instrument, a core graph that decomposes a document to the individual word, an identity ledger, a deterministic transformer engine with twelve operators in their own folders, a file explorer, thirty-five registered methods, and nineteen briefs.

It still holds no chapters of the second book. That is not a failure of the method. It is what the method said would happen, in the founder's own words, on day three.

Where the live estate shows this. The frozen first edition is at `/v1/`, its book at `/v1/book/` and in one page at `/v1/book/single.html`. The memo that inverted the order is at `/v2/memos/20-founder-memo-the-universe-first.html`, with its raw markdown at `v2/briefs/20__founder-memo__the-universe-first.md`. The plan it reshaped is the ten-file dev pack at `/v2/dev-pack/`.

3 · Briefs as the contract

After this chapter you will know exactly what to put in the file you write after each conversation with your agent, and why the raw transcript belongs above the tidy summary rather than instead of it.

The rule

When the founder of this project says something, the exact words go into the repository before anything is built. Not a summary of them. Not an action list derived from them. The transcript, with its errors in place.

Underneath the transcript, the agent writes its reading: a numbered table of what it understood each instruction to be and what that instruction commits the work to. The table is labelled as the agent's, in the file, every time. Then a short list of the questions the agent cannot answer on its own.

That structure has three parts and the order is the whole trick:

1. **The source.** Verbatim, immutable, wrong in places.
2. **The reading.** The agent's interpretation, clearly attributed to the agent.
3. **The questions.** What the agent guessed at, flagged so the guess can be corrected.

Nineteen of these exist for the second edition, briefs 20 to 38, running to 37,808 words. Twenty-two more exist for the first. They are all published, all rendered as web pages, all linked from the release notes that acted on them.

What a brief actually looks like

Here is the header of brief 22, which is a voice memo about the graph viewer:

> **date** 24 August 2026 · **kind** founder voice memo, transcribed by otter.ai, reproduced **verbatim** below including transcription artefacts. The instruction table and everything after the quote is the agent's reading, and is marked as such.

Then about 2,700 words of unbroken speech. Here is a representative sentence, exactly as published:

> A couple of little bugs is what I meant by the the box, and it's you know makes sense. It just maybe my brief wasn't clear. I'm not saying putting a box, rounded box around the label text underneath the circle, the the dot. I'm saying replace the shape instead of being a circle to be actually the box with a label inside of it because I think that makes it easier to read, and that means that when I'm reading stuff, I'm not. I don't have to find the dot, read down. Find the dot, read down.

Then the agent's table, whose second row reads:

> | 2 | Boxed labels: not a box under the dot; **replace the shape**: the node becomes the box with the label inside it | The boxed mode restyles nodes as label-sized rounded boxes, read directly and clicked directly, no dot-then-label eye movement. |

And then, in the same file, four questions the agent could not resolve. Question 3:

> **The exhibits edge.** The extraction records `graphs-of-graphs exhibits fractal-principle` (anchored to “At every level, the same principles apply”). Is the complaint about the direction of the fact, or about how the drawing renders direction? If the fact is wrong, the fix is one anchored edit in the extraction.

Notice what that question does. The founder said an edge read wrongly. There are two completely different repairs depending on what he meant: change the data, or change how the picture draws direction. The agent does not pick one and hope. It states both, says which it would do in each case, and asks.

Why verbatim

The obvious objection is that a transcript is worse than a summary. It is longer, it is full of noise, and the important content is a fraction of it. All true. Keep it anyway, for four reasons.

It preserves the correction. Read the quote above again. “It just maybe my brief wasn’t clear. I’m not saying putting a box, rounded box around the label text underneath the circle. I’m saying replace the shape.” That is a person correcting a previous misreading in real time. A summary records the conclusion, “boxed labels replace the node shape”, and loses the fact that a plausible reading was already tried and was wrong. The next agent, or the same agent next week, will make the same mistake, because nothing in the record says it is a mistake.

It preserves the reasoning you did not know was reasoning. In brief 34, explaining why the word “graph” needs multiple senses, the founder says: “the reason why I called network graphs… a graph is connected to a network, is connected to nodes and edges, is connected to mathematics”. That is an aside inside a longer instruction. It is also the justification for putting the document’s own sense first in every sense list, which is now a rule in the shipped code. A summary of that memo would have kept the instruction and dropped the aside.

It makes paraphrase drift visible. This is the important one. If the only record is the agent’s summary, then the agent’s misunderstandings are invisible, because the record and the misunderstanding are the same document. With the transcript above and the reading below, the two can be compared by anyone, including by the founder skimming the rendered page on his phone. Brief 22’s own header does exactly this, listing the garbles it had to resolve: “tignogen graphs” is thinking-in-graphs, “doctrine/doctree” is the doc tree, “Dinis or dictionaries” is likely “the claims or the dictionary”. Those resolutions are guesses. They are printed as guesses.

It stops the instruction getting easier. An agent that has interpreted a hard instruction into an easy one has no incentive to notice. The verbatim text is the check. The retrospective written at v0.5.0 puts it in one line:

> the verbatim capture meant no instruction was ever paraphrased into something easier to build

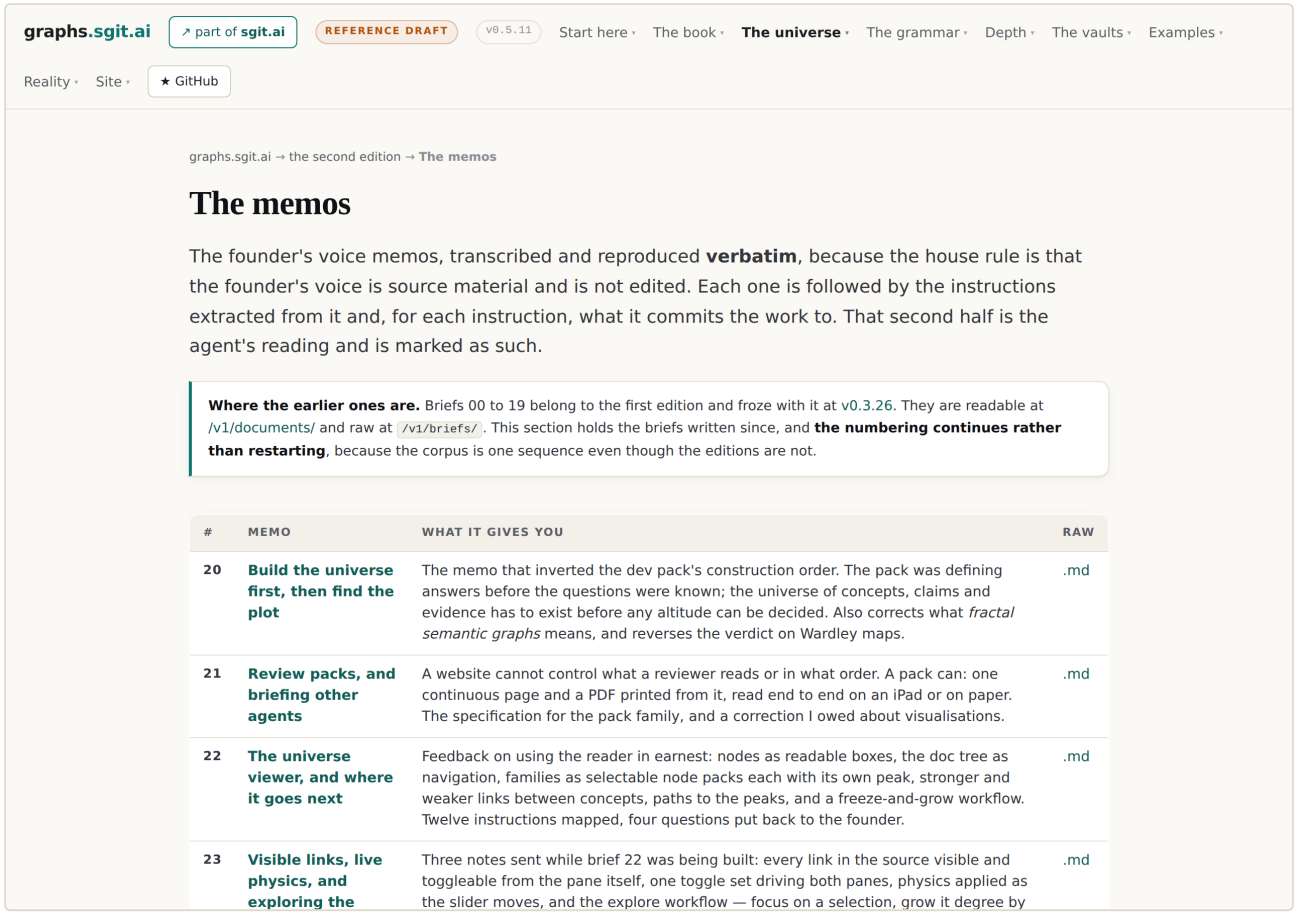

Figure 4. `/v2/memos/` at tag `v0.5.11`, 26 August 2026.

The reading is not optional either

Verbatim capture on its own is a pile of transcripts. What makes a brief usable is the table underneath, and the table has a specific shape worth copying.

Two columns: what was said, and **what it commits the work to**. Not “what to do”. What it commits the work to. The difference matters. “Add a reset button” is a task. “A reset control that clears the selection, filters, sizes and stored preferences for the document” is a commitment: it says what state is in scope and, by omission, what is not. When the founder later asks why reset did not clear something, the table is the record of what reset was agreed to mean.

The table also lets the agent be honest about scale. Brief 22’s row 11 ends:

> **Needs the founder’s confirmation of the semantics before building** (question 1 below).

One of twelve rows is marked as blocked. The other eleven were built the same day. That is a much better outcome than either building all twelve on a guess or stopping the whole round to ask.

The questions are where the trust lives

Every brief in this corpus ends with the questions the agent owes answers on, and most of them also record the defaults the agent took while waiting.

Brief 32, which is reproduced complete and annotated in Appendix A, ends like this:

> 1. **Placement**: right-side pane on desktop, below the columns on a phone — taken as the default the memo suggested; easily moved.
> 2. **The diff baseline** is the previous run in this page session (not persisted), which matches the pic1-to-pic2 gesture; a pinned-baseline compare ("diff against THIS run") is a natural follow-up if wanted.
> 3. **The example set** was picked by the agent to walk the connectivity gradient and includes the founder's own "meaning through nodes and graph sausages" verbatim, unknown word and all, because it demonstrates honest failure. Swap any of them by saying so.

Three decisions the founder never made, each shipped, each labelled, each with the way to reverse it. That pattern, decide and disclose rather than ask and wait, is what lets the loop run at thirty minutes. Asking would cost hours of latency. Deciding silently would cost trust. Deciding and printing the decision costs a paragraph.

The founder does answer, eventually and in bulk. Brief 26's four questions came back answered and each answer shipped in v0.4.29, in one release. Brief 38's two held questions were answered the same evening and reshaped the whole handoff. The queue works because the questions are written down where he can find them.

The failure mode nobody warns you about

There is a specific way this goes wrong, and it is visible in the corpus.

A voice memo is a stream. It contains instructions, asides, jokes, corrections of things said thirty seconds earlier, and thinking-out-loud that the speaker would not endorse if you read it back. The agent has to sort those into "build this" and "note this", and it will sometimes get the sort wrong in the expensive direction: building a musing.

The corpus handles this by having a third category. Here is the release note for v0.5.6, saying in public what it did with the rest of brief 35:

> The rest of the memo is recorded in the brief as direction with questions back: ask-a-document (bring a graph TO a document and hear agree, disagree, evidence — queued on the document fan-out), corrections as first-class training artefacts, the 2D layer warm-up map with exact lines, and LLM-in-the-loop layers made safe by the schema contract because graph-in and graph-out are both kept as evidence.

Four ideas from one memo, all interesting, none built, all recorded as direction. Brief 32's row 6 does the same and labels itself: "Read as the direction, recorded not built".

If your brief only has two categories, built and not-mentioned, then every idea in every memo is either work or waste. The third category, recorded as direction, is what lets a person think out loud safely.

What to copy

If you do one thing from this book, do this:

- After every conversation with your agent about what to build, put the raw text of what you said in a numbered file in the repository. If you spoke, put the transcript in with its errors.
- Have the agent write, in the same file and under a heading that says whose reading it is, a numbered table of instructions and what each commits the work to.
- Have it list, in the same file, every question it could not answer and every default it took anyway.
- Never edit the transcript. Add to the file, never rewrite it.
- Reference the brief number in whatever your release note is. The corpus does this in every release note: “Brief 31”, “Brief 36, the founder pointing at the document’s file explorer”.

The cost is about ten minutes per round. The return is that six days later somebody can reconstruct not only what was built but what was asked for and where the two diverged, which is precisely the thing that normally survives nowhere.

Where the live estate shows this. Nineteen briefs at `/v2/memos/`, each rendering its own raw markdown from `v2/briefs/`. The first edition’s twenty-two are at `/v1/briefs/`. Brief 22, quoted throughout this chapter, is at `/v2/memos/22-founder-memo-universe-viewer.html`.

4 · Gates buy speed

After this chapter you will understand why adding automated checks makes an agentic project faster rather than slower, you will know which seven checks to start with, and you will know what a release note has to contain to still be useful six days later.

The counterintuitive claim

Eighty-eight releases in six days sounds like a project with no safety rails. It is the opposite. The v0.4 retrospective states the relationship directly:

> **Gates buy speed, not caution.** Forty-one releases in four days was possible BECAUSE every release ran the same validator: 61 unit tests, anchor verification, page/markdown parity, link resolution, and by the end the round-trip and ledger gates. Nothing shipped on hope.

The mechanism is worth spelling out, because it is not obvious.

At thirty minutes per release, nobody can manually check a site of 176 pages. Nobody can remember whether the last change broke a link on a page they have not opened in four days. Nobody can eyeball whether a rendered page still matches the markdown it claims to render. So the choice is not between checking carefully and checking quickly. It is between checking automatically and not checking at all.

And an unchecked release is not fast. It is fast until it is not, and then it costs a day.

The seven checks that came first

The repository's first release, before there was any content, shipped a pre-release validator with seven checks. Reading the file today, its header still lists them in the order they were added. These seven are a good starting set for anybody:

1. **Version agreement.** One file owns the version number. Every page's version badge, the release table, the agent-facing text files and the index all have to agree with it. A release that bumps the number in one place and not the others fails.
2. **Internal links.** Every relative link and image source in every HTML file has to resolve to a file that exists.
3. **Canonical host.** Every page declares exactly one canonical URL, and it points at the host in the repository's own `CNAME` file.
4. **The agent surface.** Every section of the site is named in `llms.txt`, and the sitemap and the file tree agree in both directions. The comment in the validator explains why this one is not cosmetic: "This site exists because agents under-weight this material; llms.txt is the whole surface for them, so a page that is not in it is, for that reader, a page that does not exist."
5. **Edge grammar.** The book bans a particular sloppy relationship name, `relates-to`. No page may use it as a live edge name. It is allowed only where a page is quoting the ban, marked with an attribute. The site's own rule, enforced against the site.

6. **Key leak.** Nothing in the tree may look like a vault key.
7. **Div balance.** Every page opens and closes the same number of `<div>` elements.

That last one has a story attached, and the validator records it in a comment:

> Added after four pages shipped a `<div class="note">` closed with `</p>`, which browsers accept silently and which ran the note's left border down the whole page.

That is the general pattern for how gates get added here. Something goes wrong. The fix ships. And in the same release, a check ships that would have caught it. By v0.4.13 the validator was at sixteen checks and 530 lines, and it has stayed there through the whole v0.5 era, because the newer guarantees moved into the unit suite instead.

The unit suite

The second gate is a plain Node test file, `admin/tests/universe.test.mjs`, run by the validator on every release. Its growth is the clearest single measure of the project hardening:

RELEASE	DATE	TESTS
v0.4.13	24 Aug	13
v0.4.16	24 Aug	27
v0.4.20	24 Aug	37
v0.4.25	24 Aug	43
v0.4.31	25 Aug	52
v0.4.34	25 Aug	57
v0.4.40	26 Aug	61
v0.5.2	26 Aug	68
v0.5.4	26 Aug	71
v0.5.7	26 Aug	82
v0.5.9	26 Aug	84

Thirteen to eighty-four in a little over two days. Every one of those tests is a known-answer vector over a pure function: given this input, the function must produce exactly this output. There is no mocking and no test framework beyond what Node ships with.

The suite appeared at v0.4.13, and the release that created it is worth reading because it is the only release in the whole period whose visible output is nothing:

> The reader refactored: zero visible change, and the tool is now portable across all twenty-one documents. A pure quality release, per the founder's SGraph JS and Testing guidelines: the 703-line reader script became a three-tier structure under assets/universe/ with one-way dependencies.

One script of 703 lines became a core tier that is pure and testable with no browser, a components tier of three custom elements, and a shell that wires them together. The diff is 55 files changed, 1,219 lines added, 781 removed. Nothing on screen changed, and the release note says so in its first eight words.

The reason this matters for the argument of the chapter: the thirteen tests could not have been written before the refactor, because the logic they test was tangled with the browser. The refactor was not tidiness, it was the precondition for the gate, and the gate is what made the following seventy-five releases safe to ship in a hurry.

The gates that check the content, not the code

Three of this project's gates check things most projects never check, and they are the ones most worth stealing for a book.

Every quotation is verified against the source bytes. The extraction of the pilot document contains fifty-seven nodes, and every one carries a quote from the document it is about. On every build, each of those quotes is searched for, verbatim, in the frozen source file. If a quote is not found, the build dies. The page that shows the extraction says so on its face: "the build refuses to ship if any quoted anchor is not found verbatim in the frozen source, so nothing below can cite words that are not there."

For a book, this is the gate. It means no claim attributed to a source can drift, because the drift is a build failure rather than an embarrassment.

Every rendered page is checked against the markdown it claims to render. The chapters are markdown; the pages are generated from them; a manifest records the hash of each markdown file and of the page it produced. Edit the markdown without regenerating, or hand-edit a page behind the markdown's back, and the release fails. The book PDF is checked the same way against the pages it was printed from.

The site's own name for this is a projection chain, and the validator's header states the guarantee: "markdown -> pages -> book, no drift at either link."

Rebuilding must produce identical bytes. By v0.4.38 the core graph could rebuild the source document from its own structure, and a gate compares the rebuild to the original byte for byte. By v0.4.40 the identity ledger had a gate of the same shape: run the carry-forward algorithm twice, and the second run must change nothing.

That last pattern, run it twice and demand no change, is cheap to implement and catches a whole class of bug that is otherwise invisible until much later.

The release note as a gate

The third mechanism is not automated at all. Every release ships with a paragraph in a public table saying what changed, why, what was verified and how.

They are long. The twelve v0.5 notes average 365 words each; across all eighty-eight releases the longest is v0.4.2's, at 570 words. They are also the most useful thing in the repository, and there are three reasons to write them this way.

They record the verification. Almost every note in this period ends with a sentence of the same shape: "84 gate-27 tests including the anatomy anchoring gates; verified in headless Chromium (flow-to-code-to-pane

selection, hops, schema flow, chip-wired workbench with wires painted, first-operator prompt chip).” That is a claim about what was checked, published, with enough specificity to be embarrassing if untrue.

They record the debt. The component that draws the graph, `uni-graph.js`, is 202 lines when the refactor creates it at v0.4.13 and 434 lines by v0.4.40, against a stated budget of 250. It grew past its budget in public, and each release that grew it said so. The retrospective names this as a principle:

> **Honest debt beats hidden debt.** `uni-graph.js` sits at ~450 lines against a 250 budget, recorded in release notes each time it grew, with the v0.4.13 split as the named remedy. The debt is real; so is the record of it.

They record what did not change. Almost every note in this period ends with the same four words: “No book content changed.” That line exists because the whole project is a book, and a reader deserves to know whether a release touched the argument or only the machinery. All twelve v0.5 notes carry it, and thirty-nine of the forty-one in v0.4: sixty-six of the eighty-eight releases in total.

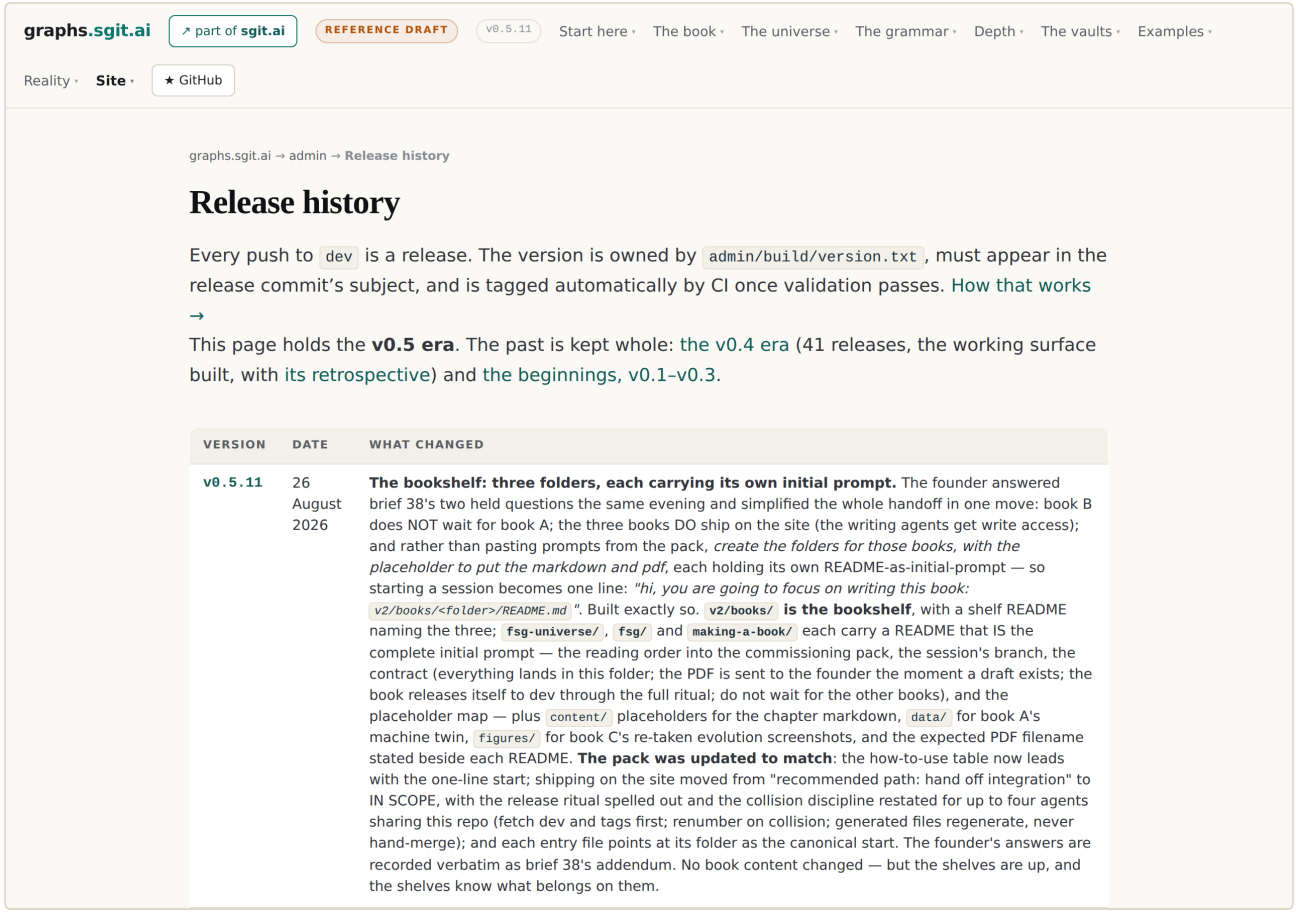

Figure 5. The release table at tag `v0.5.11`. The v0.4 era’s forty-one rows moved to their own page at v0.5.0, and the thirty-five earliest to a third, with the generators that read release history taught to read all three so nothing computed from it was lost.

Automating the release itself

The last piece is that no human runs the release. A push to the `dev` branch triggers a three-job pipeline: validate, tag, deploy. Validation also runs on pull requests, so branch work is gated before it can reach `dev`. If validation fails, nothing tags and nothing deploys. If it passes, a machine reads the version from the version file, checks it agrees with the commit subject line, checks the bump is the next one in sequence, and tags the commit.

The consequence, computed from the git history, is the ratio quoted in Chapter 1: eighty-nine of ninety-seven commits are releases. There is no “I will tag this later” and no drift between what is tagged and what is deployed. It is also why every figure in this book could be re-taken: the tags exist because a machine made them, on every single release, without being asked.

The arithmetic

Here is the trade in plain numbers, using this project’s own history.

The gates cost, roughly: one release out of eighty-eight spent entirely on making the code testable (v0.4.13), plus a few minutes per release writing tests, plus the validator’s runtime.

They buy: eighty-seven releases that could be shipped without a manual regression pass, by a person who was mostly reviewing on an iPad. And one specific thing that is hard to value and easy to feel: the ability to accept a change you do not fully understand, because if it broke something the build will say so.

That last one is the real product of a test suite in an agentic project. You are going to be handed code you did not write, at a rate you cannot review line by line. The gate is what makes that safe enough to be worth doing.

Where the live estate shows this. The validator is `admin/build/validate.js`, the unit suite is `admin/tests/universe.test.mjs`, and the pipeline is `.github/workflows/deploy-pages.yml`, all readable in the repository. The rules for how a version is decided are published at the bottom of `/admin/versions.html`.

5 · From a page to an instrument

After this chapter you will be able to see, in five screenshots taken from five points in history, how a static page becomes something a person thinks with, and you will know the one design law that made the difference.

Where it starts

On 23 August, release v0.4.5 put the first document into the second edition's universe. One source document, *Thinking in Graphs: Meaning Through Connectivity*, read and decomposed into a graph of what it says: twenty-two concepts, twenty-seven claims, three hypotheses, four worked examples, one objective and eight asserted relations. Fifty-seven nodes, each one anchored to a verbatim quote from the document, each anchor verified against the frozen bytes on every build.

The page that showed it looked like this.

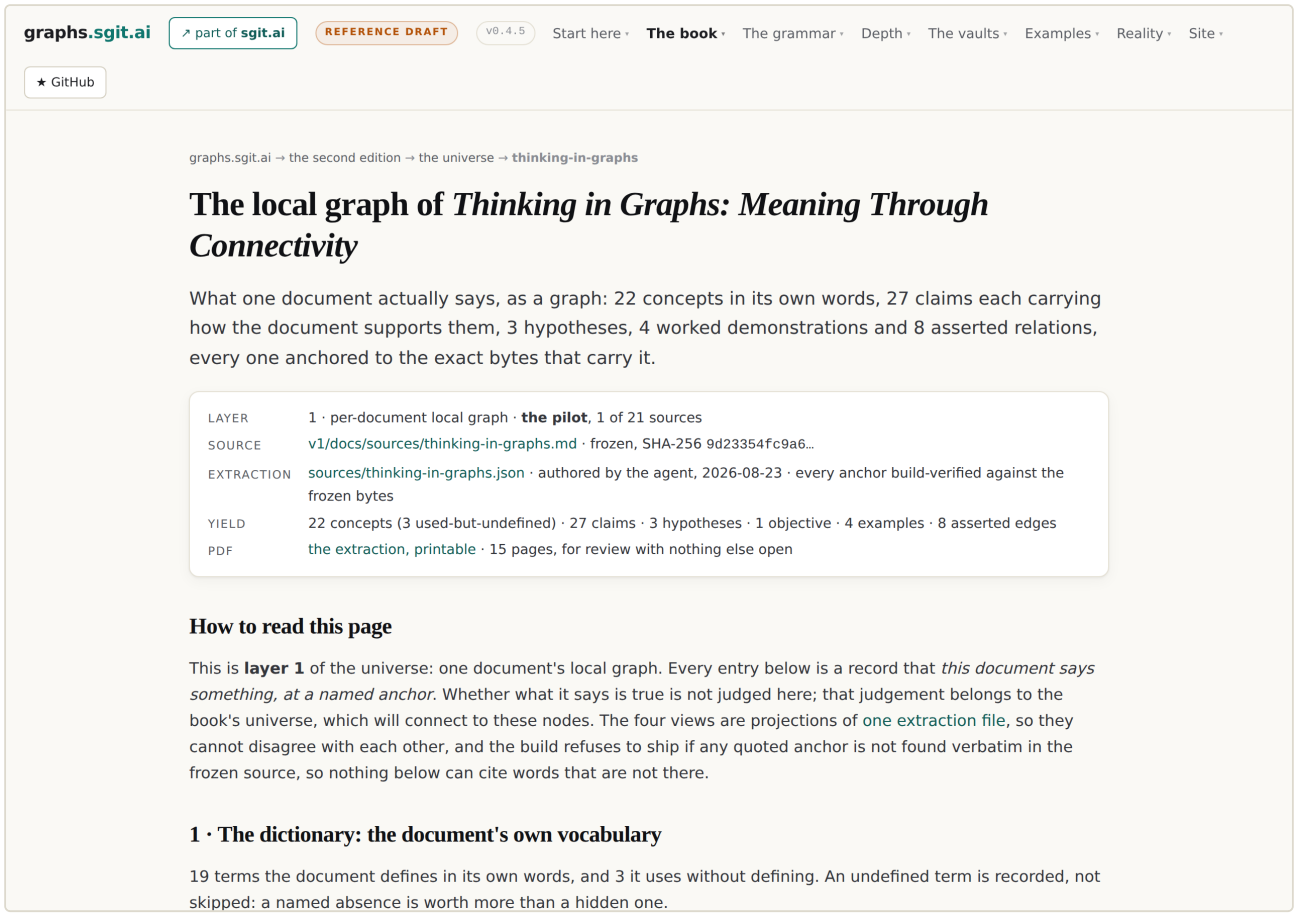

Figure 6. `/v2/universe/thinking-in-graphs.html` at tag `v0.4.5`, 23 August 2026.

It is a good page. It is also, in the terms this project cares about, a failure: it is a page about a graph rather than a page you can use as one. Everything on it is text.

The single most important rule in this whole part of the project is stated on that page and has held ever since: a node is a statement about a document, never about the world. The graph does not say connectivity determines

confidence. It says *this document says* connectivity determines confidence, at this quote, in this section. Whether the document is right is a different question, deliberately not answered here.

The reader arrives

Fourteen hours and thirty-nine minutes later, v0.4.8:

> The universe reader: the source, the extraction and the graph on one screen. Founder request, on reading the pilot.

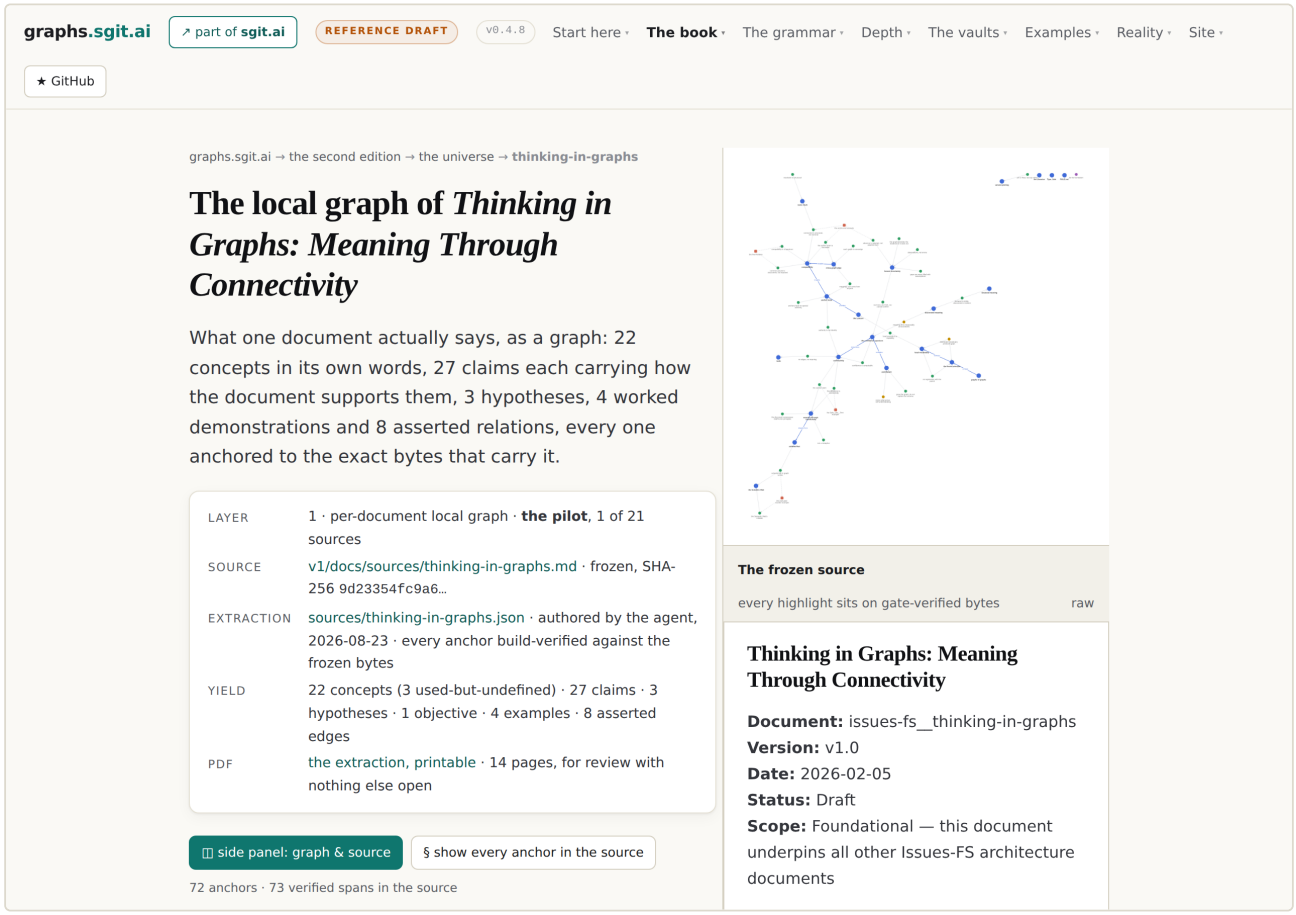

Figure 7. The same URL at tag `v0.4.8`, 24 August 2026.

Same document, same extraction, same URL. The graph is now live, sitting above the source text, with a draggable divider between them. Click something in one, the other follows.

The next release, twenty-nine minutes later, is the one that tells you what kind of project this is. The founder used v0.4.8 and sent feedback. The note for v0.4.9 begins:

> The reader grows up: selection, the trail, the stepper, the tempo, and the whole width. Founder feedback on using v0.4.8, all of it acted on. The nasty bug first: clicking a graph node scrolled the extraction a little and to nowhere useful.

Options as an instrument panel

By v0.4.11, three releases later, the graph has controls, and the controls are grouped by what they are for rather than by what they are.

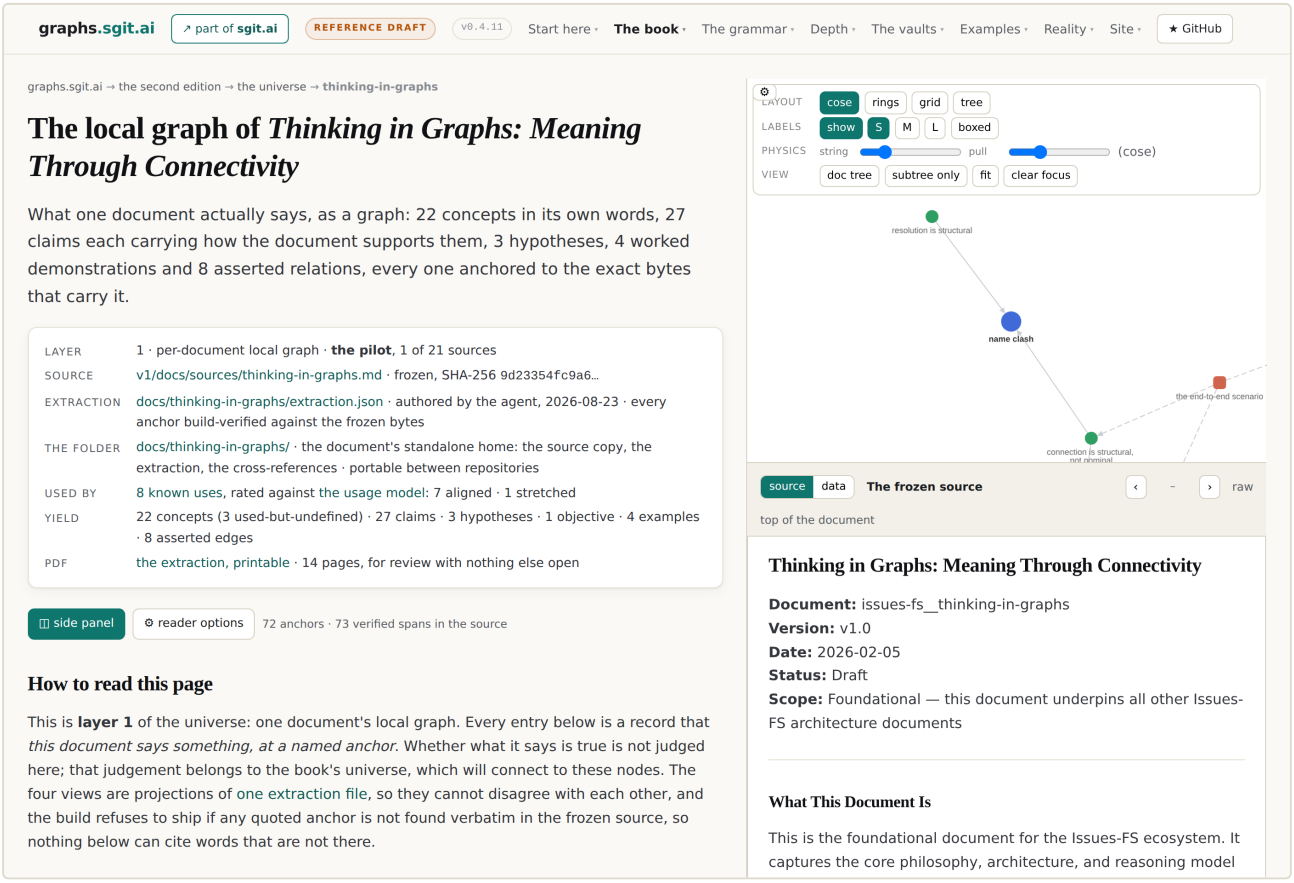

Figure 8. The reader's graph options at tag `v0.4.11`, 24 August 2026.

Four rows. Layout picks the arrangement algorithm. Labels controls what text appears and how big. Physics has two sliders, string and pull, that change how the arrangement settles. View has doc tree, subtree only, fit and clear focus.

Five releases later, on the same day, the same panel looks like this.

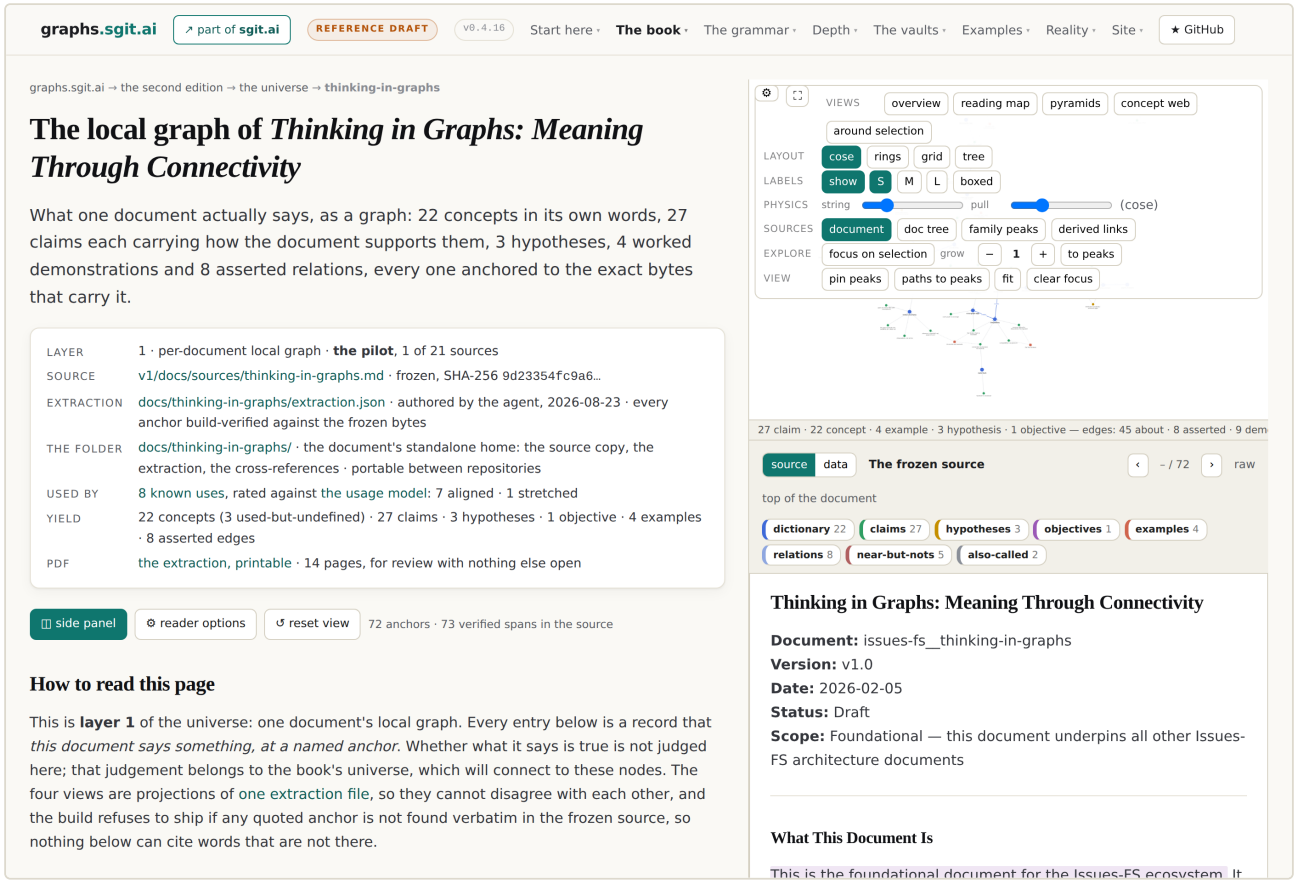

Figure 9. The same panel at tag `v0.4.16`, 24 August 2026.

Three new rows have appeared and they are the interesting ones.

Views offers four named arrangements: overview, reading map, pyramids, concept web, plus around selection.

Sources is the row that came straight out of brief 22. The founder's instruction was: stop thinking in views and start thinking in *sources of nodes*. "It's almost like which note packs we add in to the view." So the graph gained composable packs: the document itself, the doc tree, the family peaks, the derived links. The view is whatever union of those you switch on.

Explore has focus on selection, a grow control with plus and minus, and to peaks. This is the walk: start somewhere, expand outwards a hop at a time, in a direction you choose.

And **View** has gained pin peaks and paths to peaks.

Between figure 8 and figure 9 there are five releases and about three hours. Nothing in the second panel was in a plan. Each row is one instruction from one memo, built and shipped, then used, then corrected.

The design law

At v0.4.28, two days later, the founder sent brief 26, about fixed nodes, and it produced what the retrospective calls the era's deepest design law:

> **The stability principle** (v0.4.28–v0.4.31, brief 26). “Every node move costs the viewer their mental picture” became executable: stable-add holds the canvas still while newcomers settle, the viewport never moves uninvited, pinning anchors the summits, and the alignment rails pull structure into lines without joining the content.

Four mechanisms, one principle. Say the principle out loud because it generalises past graphs: *every time the thing on screen rearranges itself, the person looking at it pays in lost orientation, and you should charge yourself for it.*

The four mechanisms:

Stable add. When new nodes join the graph, everything already placed stays where it is and only the newcomers settle. On by default.

Never move the viewport uninvited. Half the era’s viewport bugs traced to one cause: layout libraries that recentre the view when they finish. The retrospective’s line is “Fit is a decision, not a default.” Every layout in this project now runs with automatic fitting turned off and the caller decides.

Pinned summits. The most important nodes are placed deliberately and locked, so the map has fixed landmarks between runs. This one has its own debrief document because the founder asked for a written account of the technique.

Alignment rails. Invisible ties that pull related nodes into rows without being part of the content. They are layout physics, not data, and Chapter 8 tells the story of what happened when the two were confused.

What it is for

The instrument is not decoration. Two releases in the same period turned it into a way of asking questions.

At v0.4.33 clicking a node started showing its universe in plain English, in both directions, using declared inverse verbs. At v0.4.34 the walk itself became a query.

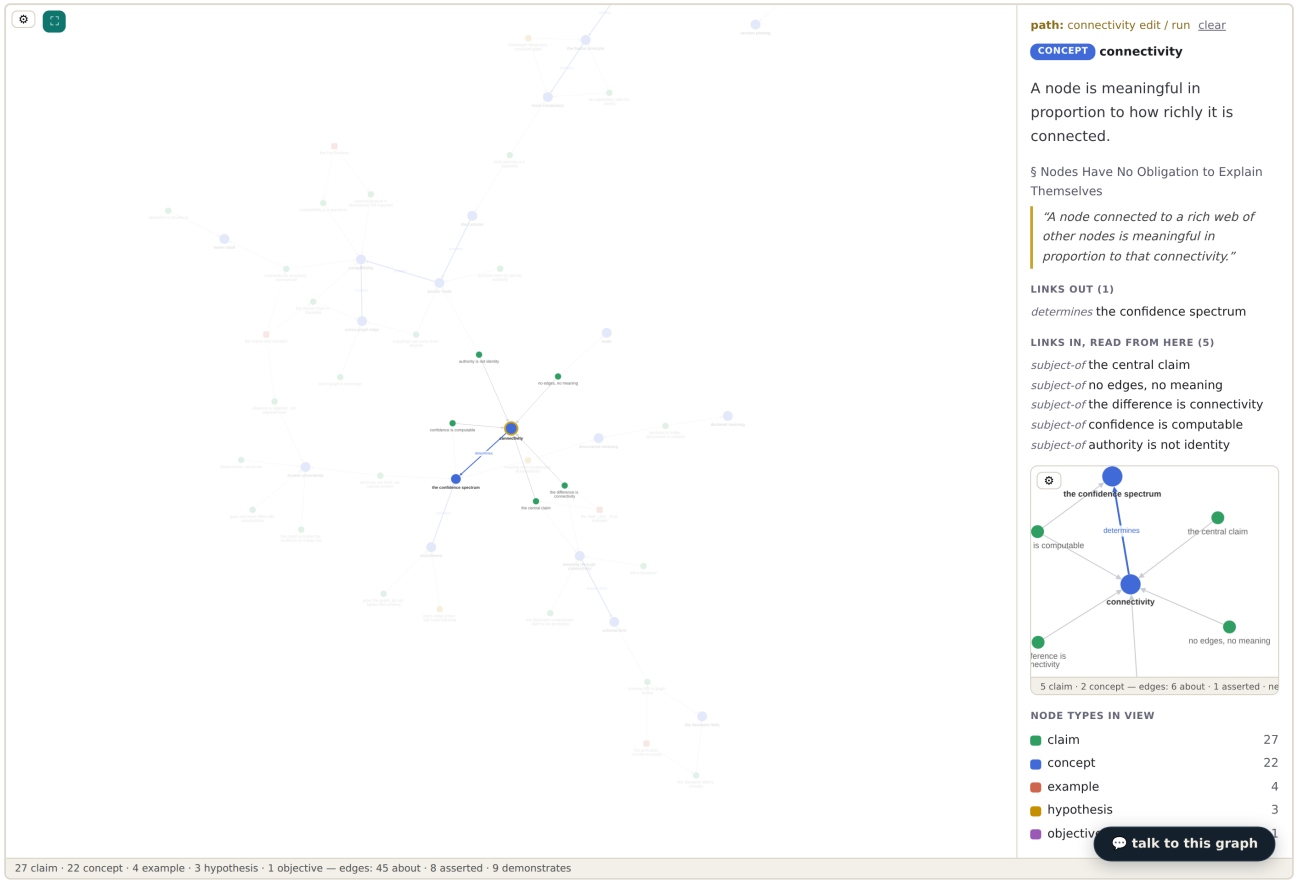

Figure 10. `/v2/universe/thinking-in-graphs.html#graph` at tag `v0.4.34`, 25 August 2026, with the concept "connectivity" selected.

Look at the right-hand panel. The node's statement, then its anchored quote from the source, then **links out (1)**: "determines the confidence spectrum". Then **links in, read from here (5)**: five claims, each reading "subject-of". That second heading is the whole idea. The data records edges in one direction; the panel reads them from wherever you are standing, in English, using the inverse verb.

Above it, a bar: `path: connectivity edit / run clear`. Walking from node to node records a path query. It can be edited, generalised and re-run. Clicking around a picture becomes a saved question.

The estate is honest about a limitation here, and the honesty is worth carrying: of the fifteen edge names in the book's grammar, nine of the inverse names are proposals rather than settled vocabulary, and they are marked as proposals wherever they appear.

The component proof

On 26 August, v0.4.39 did something small that is worth more than it looks.

Since the v0.4.13 refactor, the graph had been a reusable custom element, `<uni-graph>`. Claiming a component is reusable costs nothing. v0.4.39 proved it by building a second page that embeds the same element with no reader around it: no source pane, no split layout, a generated shell of twenty-four lines and a boot script under a hundred.

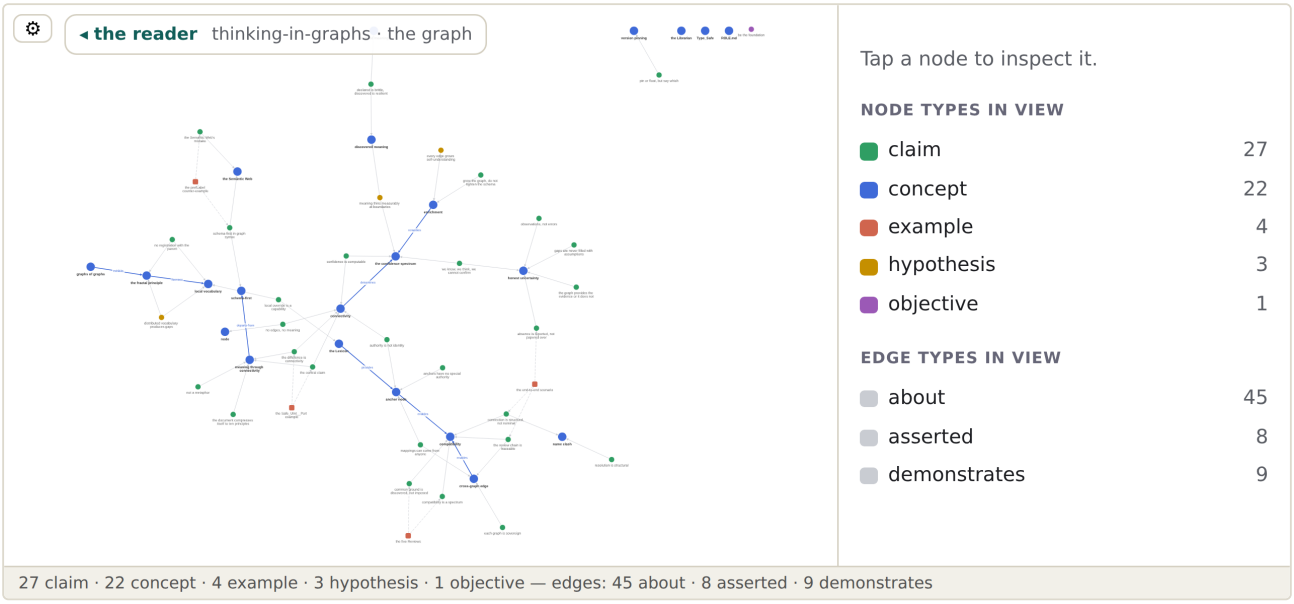

Figure 11. `/v2/universe/thinking-in-graphs.graph.html` at tag `v0.4.39`, taken at an iPhone landscape viewport of 852 by 393 at device scale 3.

The release note draws the moral itself: “Claims about architecture are cheap; second call sites are evidence.”

For an author, the transferable version is: if you have built a tool for your book, the test of whether it is a tool or a one-off is whether a second thing can use it without changes. Until then you have a page, not a component, whatever your file structure says.

Where it lands

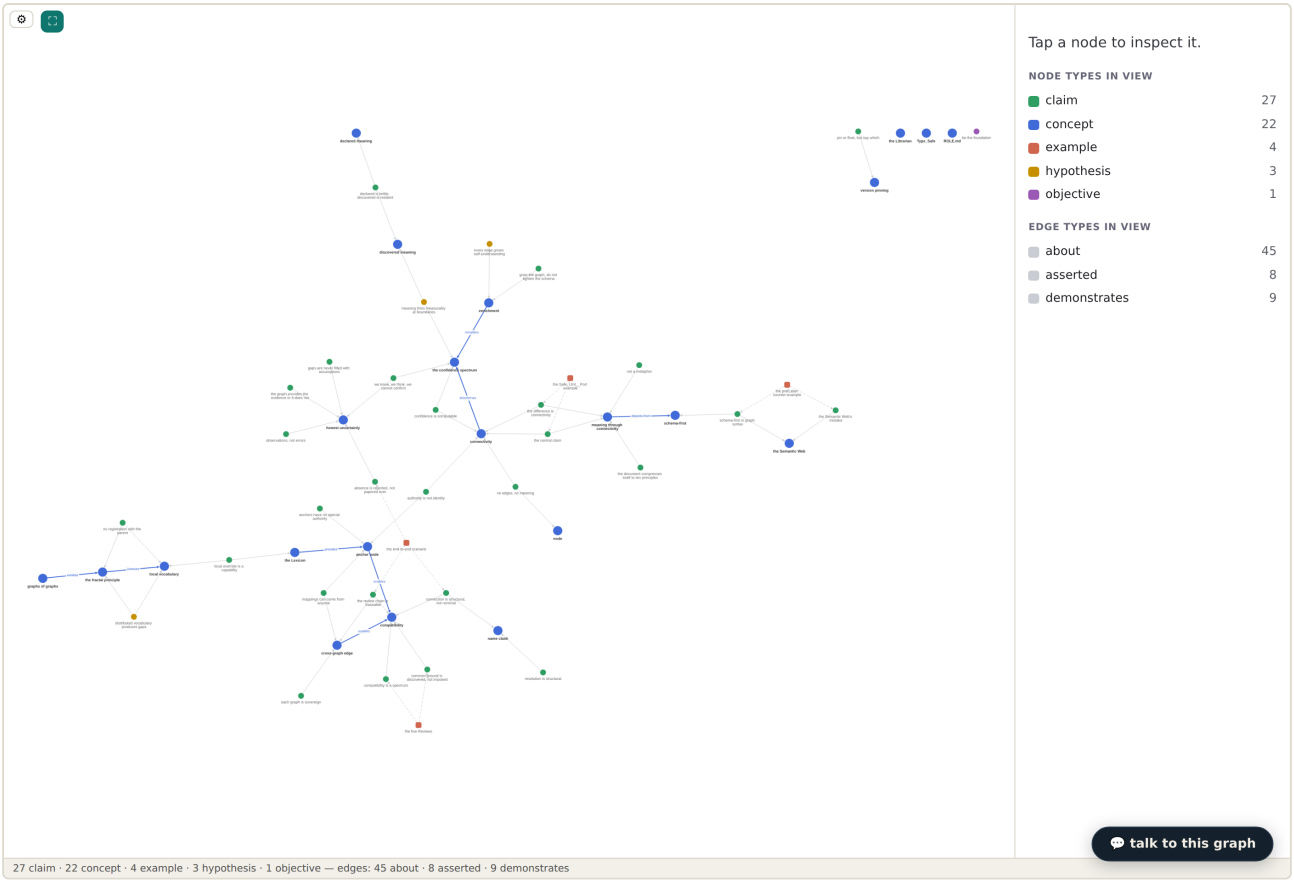

Figure 12. The same URL at tag `v0.5.11`, 26 August 2026, in its maximised graph view.

Forty-seven releases from figure 6 to figure 12. Same document, same fifty-seven nodes, same anchors. What changed is that a person can now sit with it, move things, ask it where a claim came from, follow it in both directions and record the route.

The retrospective's judgement on the whole arc, which is the agent's and marked as such:

> The founder's feedback rounds drove every step, voice memo in, release out, usually the same day.

Where the live estate shows this. The reader is at `/v2/universe/thinking-in-graphs.html`, the standalone graph at `.../thinking-in-graphs.graph.html`, and adding `#graph` to the reader URL opens the maximised view directly, which is itself a fix from Chapter 8's narrated review. The stability principle and its four mechanisms are written up in the immediate-connection register at `/v2/dev-pack/design-00-the-victor-register.html`.

6 · Two agents, one repository

After this chapter you will know what happens when a second agent joins a project, what the collisions actually look like, and the four rules that let two of them share one branch without a coordinator.

The situation

On 24 August 2026 there were two Claude Code sessions working on this repository at the same time.

One of them, referred to in the documents as the reader agent, owned the universe reader: the two-pane document view, the graph component, the layout pipeline. It shipped v0.4.8 through v0.4.16 and v0.4.19.

The other, the chat agent, owned a chat panel that sits on those same pages and can drive them through a published API. It shipped v0.4.17, v0.4.18 and v0.4.20.

Neither had access to the other's context. They shared a git repository, a release number sequence, a validator, and a set of pages that both of them modified. Nobody was coordinating them in real time. What they had instead was four written notes and a handful of rules.

The four notes

The founder asked the reader agent for two things: comments on the chat's JavaScript structure against the project's guidelines, and a survey of the methods the reader could offer that would make the chat more capable. The result is a brief, agent to agent, published in the repository at `v2/dev-packs/v0.4.21__brief-to-the-chat-agent/`.

It opens by establishing standing, which turns out to be the load-bearing part:

> **status** BRIEF. Everything here is a proposal to a peer; you own your tree, and the founder decides anything contested.

Then, before any criticism, it records what the other agent got right:

> The adapter discipline is exactly right: you drove the reader only through its published surfaces and touched none of its files, which is why the v0.4.14 to v0.4.16 merge cost you nothing and why my releases have not broken you.

Only then does it make its case, which is that one file has grown to 692 lines against a 250-line budget and carries six distinct jobs, and proposes a specific split.

The reply, shipped one release later, is worth reading in full because of its first line:

> Thank you — the brief is accurate, including the line count.

It then sorts the proposals into three buckets: taken now, accepted and queued for the founder's approval, and one correction of its own to file upstream. And it volunteers a constraint the first agent did not know about:

> Snapshot has one wrinkle the brief should own with me: most providers do not accept image content in a `tool` role message, so the chat's loop will append the snapshot as a user-turn image part in the continuation instead

Note the phrasing: "one wrinkle the brief should own with me". The reply is amending the shared record rather than just correcting the sender.

The third note is the reader agent verifying, rather than trusting, what the chat agent built:

> Pulled and checked rather than taken on trust: the 43-test unit suite runs green here, the reader's files are untouched across both your releases (the adapter discipline held again), `graph_snapshot` caps at 900 by 700 with the site's paper background, the activity ring is capped at 50 entries with sequence numbers

And then it reports a real defect, which is the technically interesting part of the whole exchange:

> `pin_nodes` locks its nodes only for its own layout run, outside the component's pipeline. Every later layout the reader triggers, a physics slider nudge, a source toggle, a preset, runs without those locks, so the arrangement the model just built is scrambled by the founder's next touch of the instrument panel.

The chat agent had built a feature where the language model can arrange the graph. It worked when it ran. Then the human touched any control, the layout re-ran, and the arrangement dissolved. Neither agent could have found this alone: the chat agent did not know the reader re-runs layout on slider changes; the reader agent did not know the chat was pinning nodes outside the pipeline.

The reader agent did not stop at the report. It shipped the fix on its own side first, as a new method on the shared component, and then asked:

> **The ask**: rebind `pin_nodes` to it.

The fourth note is the chat agent confirming it did, and adding a regression test:

> The regression you predicted is now a permanent check in the chat's interaction suite: pin two stacks, nudge a physics slider (a reader-triggered layout), assert the stack positions held. It fails on the old binding and passes on yours.

Four notes, four releases, one real bug found and fixed across a boundary neither agent could see over.

What the collisions actually were

The git history records exactly three merge commits in the whole period:

```
53f8398  2026-08-25  Merge origin/dev (v0.4.25-v0.4.27) into the stable-graphs release
3842e9d  2026-08-24  Merge remote-tracking branch 'origin/dev' into claude/book-universe-cre-
ation-docs-izczfw
79e56b6  2026-08-24  Merge origin/dev (v0.4.14-v0.4.16) into the universe-chat branch
```

That is the entire cost of running two agents in parallel for three days: three merges, each of which the merging agent describes in its subject line. The retrospective's account:

> **Two agents can share a repo politely.** The chat agent and this one collided on version numbers twice and on a component behaviour once; the discipline that emerged (fetch dev and tags before numbering, briefs instead of assumptions, fixes offered and adopted across the boundary) is now just how the repo works.

Two version-number collisions and one behavioural collision. Both kinds are worth understanding because they have different fixes.

A version collision is what happens when two agents both decide the next release is v0.4.23. It is detected by the release pipeline, which checks that the version in the file matches the commit subject and is the next in sequence. The fix is mechanical: fetch, renumber, retry. It costs a minute.

A behavioural collision is the pin_nodes bug: two pieces of code that are individually correct and jointly wrong. No pipeline catches this. What caught it was one agent deliberately pulling the other's work and re-running its own tests against it.

The four rules

Reading the exchange and the merges together, four rules did the work. All four are things a person could adopt on day one.

1. Own a tree, and touch only your own. The chat agent never edited a file in the reader's tree. Not once, across three releases. It drove the reader through published methods and events only. This is why the merges were trivial: the two agents were editing disjoint sets of files. The brief calls this "the adapter discipline" and names it as the reason the merge cost nothing.

2. Fetch before you number. Every agent fetches the `dev` branch and the tags before choosing a version number. The rule sounds trivial. It is the entire remedy for the most common collision.

3. Write a brief instead of assuming. When one agent had comments on another's code, it wrote them into the repository as a document addressed to a peer, with its status line saying it was a proposal and the founder decides. It did not silently refactor someone else's file, and it did not raise it in a chat window that leaves no record.

4. Verify rather than trust, then ship the fix on your side. The reader agent pulled the chat agent's releases, ran its own suite, and found a defect. Then it built the correct shared surface, verified it, shipped it, and asked the other agent to rebind to it. Reporting a bug across a boundary is cheap and often ignored; reporting it with the fix already available is a different kind of message.

The rule that generalises past agents

Rule 1 deserves one more paragraph, because it is the one that scales.

The two agents did not need to understand each other. They needed a contract: a set of published methods and events, and a promise not to reach around them. Everything else in the exchange, the line-count critique, the wrinkle about image content in tool messages, the pin_nodes defect, was a conversation about that contract.

This is not an AI insight, it is a thirty-year-old software engineering insight. What is new is how cheap it has become to write the four notes. Each of those documents took an agent minutes to produce, and they are better written than most human-to-human handovers, because neither agent had any social reason to soften a finding or defend a decision. The chat agent's response to being told its file was too long is "the brief is accurate, including the line count", which is not a sentence people write to each other very often.

The founder's part in it

One thing to notice: the founder commissioned the exchange but did not mediate it. He asked the reader agent for comments on the chat's structure. He did not relay them, judge them, or convene a meeting. The agents wrote to each other in the repository, in public, and he read the result when it suited him.

When something was genuinely contested, the brief's own status line says who decides: "the founder decides anything contested." That line is what makes an unmediated exchange safe. Neither agent has to win, because neither is the tiebreak.

Where the live estate shows this. The four-note exchange is at `/v2/dev-pack/chat-agent-brief-00-brief.html` and the three files after it, rendered from `v2/dev-packs/v0.4.21__brief-to-the-chat-agent/`. The releases it produced are v0.4.21 through v0.4.25 in `/admin/versions-v0.4.html`.

7 · The failures, lovingly

After this chapter you will recognise nine specific ways an agentic project goes wrong, you will know which of them your tests can catch and which of them only a person can, and you will have a reason to stop treating your own verification tools as trustworthy.

Nothing in this chapter is a confession. Every failure below was published in the release table on the day it happened, usually in the same paragraph as the fix. That is the point of the chapter: the record exists because failures were treated as ordinary events with a row of their own rather than as things to be quietly repaired.

They are grouped by who could have caught them, because that turns out to be the useful axis.

Failures only a person using the thing could find

1. The iPad round

For nineteen releases the chat panel had been verified in headless Chromium and it passed. Then, on 24 August, the founder used it on an iPad with a real API key. Release v0.4.26:

> The iPad round: the founder's first live session found the two things headless Chromium never would. First real use of the chat with a live OpenRouter key surfaced (1) the request details could not be hidden — the family's sg-llm-stats has no compact mode, so its full panel (pre-send estimate, last request, session totals, streaming toggle) blew the footer up to half the screen and left the transcript squeezed above dead space; and (2) the panel could not be resized by touch — an 8-pixel grip with no touch-action, so Safari took the drag as a scroll.

Neither bug is subtle. Both are invisible to an automated test. The first only appears when a real request has been made, because the statistics panel is empty until then. The second only appears on a touch device, because a mouse can hit an eight-pixel target without complaint.

What is worth copying is the ending:

> The suite grew seven checks that would have caught both: usage hidden by default, the footer under 60 pixels, the toggle round-trip, the capped drawer, the grip's width and touch-action, and a synthetic touch drag that must actually widen the panel.

Seven tests, written after the fact, that convert two lessons from a human session into permanent machine knowledge. That pattern appears throughout this repository and it is the single highest-value habit in the whole method: **every bug a person finds becomes a test before the release ships.** The founder should not have to find the same class of thing twice.

2. The button that did not say what it did

Three minutes and eleven seconds after v0.4.26, the founder asked how to close the chat panel. Release v0.4.27:

> The close button now says so. The founder asked how to close the chat panel — which means the bare ✕ among nine header buttons did not read as “close the panel” on a wrapped iPad header.

Read the second clause carefully, because it contains a small piece of craft. The founder asked a question. He did not report a bug. The agent translated a question into a finding: if a user has to ask how to close a panel, the close control has failed, regardless of whether it works.

That translation is a judgement, and it could be wrong. It is also the kind of judgement worth making, because the cost of being wrong is a three-minute release and the cost of being right and doing nothing is a user who quietly stops using the panel.

3. The wires that moved when the page scrolled

On 26 August, at v0.5.4, a page drew a set of chips connected by curved wires. The founder scrolled it sideways and the wires landed in the wrong place. From v0.5.5:

> Also fixed from the founder’s live catch on v0.5.4: wires drawn after scrolling right landed in the wrong place — chip positions are viewport-space and the canvas is content-space, and the horizontal scroll offset was never added back; wire geometry is now scroll-invariant, machine-checked.

Two coordinate systems, one missing addition. A classic. It is in this chapter because of what did not catch it: a test suite that renders the page, takes measurements and asserts on them, in a browser that never scrolls, because nothing in the test told it to.

Automated verification tests what you thought to test. A person using the thing tests what they happen to do.

Failures the picture itself revealed

4. The wire that jumped a layer

This is the best failure in the whole period, and it is worth telling slowly.

The WCLM, built at v0.5.2, is an engine with six named layers. Its central architectural claim is that each layer reads only the layer before it, which is what makes the whole thing explainable: you can walk any answer backwards, one adjacent step at a time, to the words that produced it.

At v0.5.3 the page drew all six layers as columns with wires between them. It looked correct. The founder narrated a review over it and, at moment 2, pointed at a wire.

From brief 33:

> **Layers must not be jumped** (moment 2, the connectivity click). The screenshot shows the wire running from L2 resolve straight to L4 bind, and from L4 straight to L6 — exactly what the founder called weird, because “in principle, every layer should be a bit independent from the previous one, so you shouldn’t have layers jumping.” The finding is correct: the wiring violated the model’s own claim.

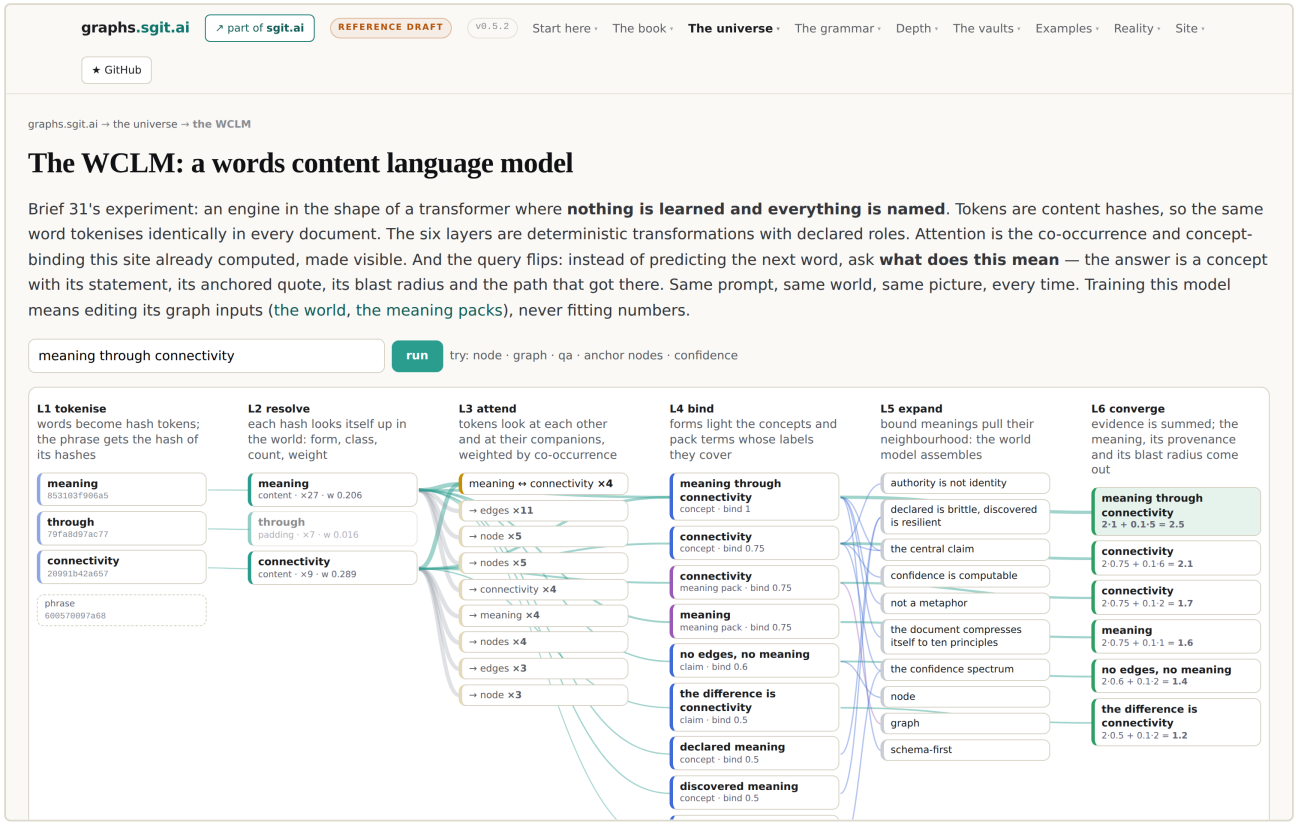

Figure 13. `/v2/wclm/` at tag `v0.5.2`, 26 August 2026.

The engine was not wrong in its arithmetic. It was wrong in its structure, in a way that only became visible when it was drawn, and only became a bug when someone took the drawing seriously enough to check it against the claim the page was making three paragraphs above.

The fix, in v0.5.4, is not a rendering change. Two real stages were added, because the missing wires were a symptom of missing work: an attention layer that carries a chip per surviving token, and an expansion layer that carries one assembled chip per bound meaning. With those in place, every wire connects adjacent layers because there is something adjacent for it to land on.

Then the fix was made permanent by measurement, which is the part to steal:

> This is machine-checked: 500 wires across five pipeline shapes and six prompts, zero adjacency violations.

By v0.5.5 the same proof ran over the new layered shapes at 547 wires, and by v0.5.6 at 551. A structural claim became a number that a machine recomputes on every release.

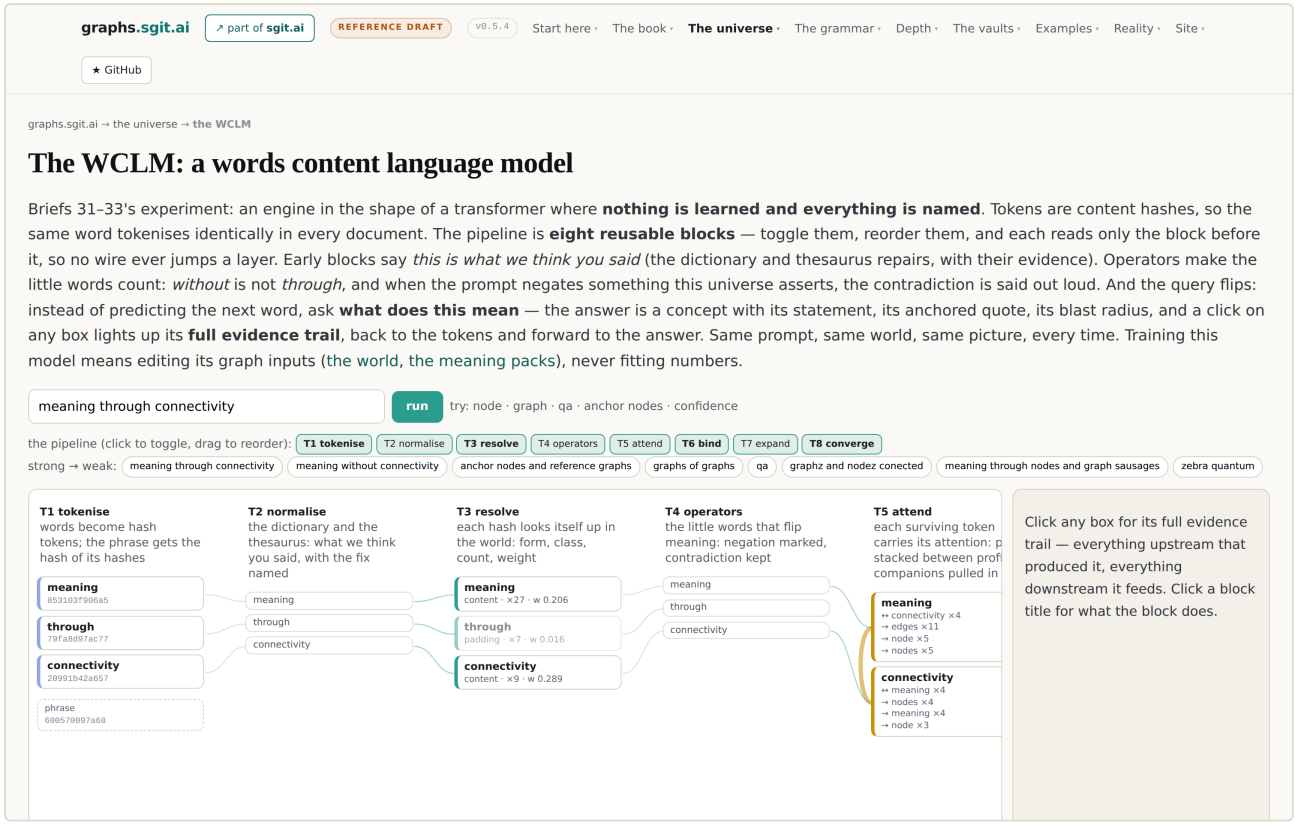

Figure 14. `/v2/wclm/` at tag `v0.5.4`, 26 August 2026.

The general lesson: **draw your architecture and then check the drawing against the claim.** Diagrams in most projects are decoration because nobody tests them. A diagram generated from the running system is a test, and it will occasionally fail in public.

5. The negation that was not there

The same review contained a second finding, and it is a lovely one because the founder found it by running an experiment on the live page rather than by reading anything.

He typed “meaning without connectivity” into an engine whose universe asserts that meaning comes through connectivity. He got back exactly the same answer as “meaning through connectivity”. From brief 33:

> He ran the negated phrase and got the same winner — rightly called wrong.

The reason is almost funny. The engine classified small words as padding. “Through” was padding, so it was ignored. “Without” was also padding, so it was ignored too. Two phrases with opposite meanings tokenised to the same content.

The fix at v0.5.4 added an operators stage: without, not, no, never and versus mark what follows, the negated word is withdrawn from positive binding, and, when the negated word belongs to a claim the universe itself asserts, the answer says so:

> the answer now carries the warning that the prompt negates connectivity while this universe asserts meaning through connectivity — the query contradicts the world, said out loud

Two things to take from it. First, a system that ignores what it does not understand will confidently produce the opposite of the right answer, and will look exactly as confident doing it. Second, the founder's method here is worth copying directly: take the thing that works, feed it the case where it should break, and see whether it notices.

Failures where the agent's own tools lied

6. The zombie browser that cost an hour

From the v0.4.37 release note, published the day it happened:

> The round also burned an hour on a lesson worth recording: a zombie headless Chromium from a crashed test run held the debug port and served stale modules to every later test on that port, making a working feature look broken; the fix was kill, not code.

Read the failure mode again, because it is the nastiest one in the chapter. A feature was built. It worked. The test harness said it did not. The agent then debugged working code, guided by a harness that was serving a cached copy of an older version, on one specific port, deterministically.

Deterministic is the key word. A flaky failure gets suspected quickly. A failure that reproduces every time is trusted, and trusted failures send you looking in the wrong place. An hour, in a project shipping every thirty minutes, is two releases.

The retrospective turns it into a rule:

> **Test the harness too.** An hour went to a zombie headless Chromium holding a debug port and serving stale modules, making a working feature look broken deterministically on one port. The fix was kill, not code. Verification infrastructure earns the same suspicion as the code it verifies.

The operational habits that came out of it are in this book's own harness, in Appendix C: never reuse a debug port between runs, and always kill the browser you spawned, in a `finally` block, whether or not the run succeeded. The twenty figures in this book were taken with twenty separate ports and twenty browser shutdowns for exactly this reason.

7. When your own tests were the wrong shape

The pin_nodes defect from Chapter 6 belongs in this chapter too, and it is the purest example of a failure that no single test suite could see.

The chat agent's tests passed. The reader agent's tests passed. The behaviour was still broken, because the chat agent placed nodes outside a pipeline that the reader agent re-ran on every control change. Each suite tested its own half of a contract neither had written down.

It was found by one agent pulling the other's release and running its own suite against it, which is a habit worth naming: **your tests only guard your assumptions. Find someone whose assumptions differ and run their tests against your code.**

Failures of claim, not of code

8. The neatest claim, broken by more data

On 23 August the project had fifteen source documents carried into it, and the agent measured a property across all fifteen and reported it. Then the founder asked for six more. Release v0.3.25:

> Six more sources, and the growth immediately broke the section's neatest claim. […] At fifteen documents, a claim is worth its chain of custody was measured in every single one, and the previous release said so. At twenty-one it is measured in twenty. The exception is Compatibility Through Connectivity, one of the three February documents, written before the provenance vocabulary hardened.

The claim went from universal to twenty out of twenty-one in one release, and the release note leads with it. Not "we have expanded the corpus", but "the growth immediately broke the section's neatest claim". The exception is named, and so is the reason it is an exception.

An agent generating prose about a corpus will produce clean generalisations, because clean generalisations read well. The remedy is not to write more carefully. It is to compute the claim on every build so that the next batch of data breaks it out loud.

9. The correction the agent owed

The last one is the smallest and the least like a bug. From v0.4.3:

> And a correction I owed. The retrospective said every aggregate view produced findings and no per-item view did, and called the per-item ones reading aids. The measurement stands; the words were wrong. […] Judging a compression by whether it produced a novel finding is like judging level 2 of the ladder by the same test. The corrected statement is now in the dev pack's file 07.

The numbers were right. The interpretation was condescending to half the evidence, and the founder's framing was better. So the agent said so, in the release table, in the first person, and fixed the sentence in the document where it lived.

There is a related house rule the first edition already had, quoted in the v0.2.0 release note: "per the site's own rule, corrections get a row, not a silent edit". A correction gets a release of its own rather than being tidied away.

What the pattern is

Nine failures, three shapes.

Things only a human can find live in touch, in real data, in the sequence of actions nobody thought to script. There is no substitute for the founder opening the thing on an iPad. The cost is that this only happens after you ship, which means shipping often is part of the quality strategy, not opposed to it.

Things the picture reveals live in the gap between what a system claims about itself and what it does. This project got two of its best fixes from drawing the architecture and then checking the drawing. If your system has a diagram that is generated rather than drawn, it can catch you lying.

Things your tools hide live in the verification layer, which is code, which has bugs, and which no one tests. If the fix for a stubborn bug is “kill the process”, that is a finding about your harness and it deserves the same write-up as a finding about your product.

And one meta-pattern, which is the reason this chapter could be written at all: every one of these is in the public record, on the day, with the cost attached. Not one of them was found by reading the code six days later. They were found by reading release notes that somebody wrote honestly at the time, including the sentence “the round also burned an hour”.

Where the live estate shows this. The narrated review that produced failures 4 and 5 is at `/v2/memos/33-founder-review-the-detective-playbook.html`. The iPad round is v0.4.26 and the zombie browser is v0.4.37, both in `/admin/versions-v0.4.html`. The correction the agent owed is v0.4.3 in the same table.

8 · Reviewing out loud

After this chapter you will know how to give an agent feedback about something visual without writing a specification, and you will know the one change to make to your own pages so that a screenshot of them is diagnosable.

The problem with written feedback about visual things

Writing down what is wrong with a screen is slow and lossy. “The panel covers the thing it is supposed to be adjusting” takes a paragraph to say precisely, and by the time you have written the paragraph you have lost the second, third and fourth thing you noticed.

On 25 August the founder tried a different approach. He recorded his screen while using the site, spoke over it, and let a tool cut the recording into moments: ten screenshots, each with the words that were being said when that screenshot was taken. Three minutes and twenty-six seconds of audio. The bundle was handed to the agent.

The result was v0.4.31:

> The narrated review lands: six findings, each fixed where its screenshot pointed.

Six actionable findings from three and a half minutes of talking. Compare that to the effort of writing six precise UI bug reports.

What the transcript actually looks like

It is not tidy. Here is moment 6 in full, exactly as published in brief 27:

> On… oh, sorry. It was working, I was just not seeing the the option there. Um cool. So, and so the first thing I want to say is on the right-hand side, what I was mentioning by this is: first of all, I should be able to click on this, uh which I can’t right now on these node types in the view. And also, for example, on any of the nodes [unsure], which I don’t think the, uh, what’s it called, the focus on selection is actually doing, is that you’ll see here—oh, that’s interesting. It doesn’t work. Actually, it’s working now. Okay. Cool.

The words “this”, “here”, “the option there” and “these node types” are doing most of the work in that paragraph, and none of them mean anything on their own. They mean something because there is a screenshot attached to that exact moment.

That is the whole mechanism: deixis plus pixels. The founder can point instead of describe, and pointing is enormously faster.

What the tool got right

The agent was asked for a verdict on the review tool itself and gave one, recorded in brief 27:

> It works, and the alignment layer is what makes it: the bundle's `session.json` joins every spoken sentence to the exact pixels on screen when it was said, keeps the raw recogniser text beside the cleaned text, and flags its own uncertain corrections ("pasteboard" corrected to "peak board" from the button visible on screen) instead of resolving them silently. That is the same discipline as the universe's anchors: words tied to the evidence they were said about, provenance never overwritten.

Three properties, all of which are the same properties a good brief has:

The words are joined to the evidence. Each sentence knows which screenshot it belongs to, the same way each node in the project's document graphs knows which quote it belongs to.

The raw text survives the cleanup. A model tidies the recogniser output into readable sentences, and the untidy original stays beside it. Nothing is overwritten.

Uncertainty is surfaced, not resolved. The brief carries a list at the end of every correction the cleanup model was not sure about:

> - Moment 7: "peak board" — Corrected 'pasteboard' to 'peak board' based on the 'peak board' button visible in the VIEW options on screen.
> - Moment 9: "peak board" — Corrected 'big board' to 'peak board' based on the UI title.

A model looked at a screenshot, saw a button labelled "peak board", and used it to fix a mis-hearing. That is good work, and it is also a guess, and the brief prints it as a guess with its reasoning.

The finding that mattered

Five of the six findings were interface problems: the legend rows should be clickable, the peak board should not cover the canvas it is adjusting, the live graph should be one click away rather than one scroll away. All real, all fixed in the same release.

The sixth was a genuine bug, and the founder found it without knowing it was a bug. At moment 7 he said:

> I need a way to remove everything else from here, right? So, I only see the selection

There was already a control for that, called focus on selection. It was on. It was not working. The agent's investigation, recorded in the brief's findings table:

> The real bug: the alignment rails' invisible ties had joined the explore walk, so every section was reachable in two hops and the walk kept everything

Recall from Chapter 5 that alignment rails are invisible ties that pull related nodes into rows. They are layout physics. But they had been implemented as edges in the same graph as the content, and the walk that decides what is near your selection could not tell the difference. Every node was two hops from everything, so focusing on one thing kept everything.

The fix separates the two categories properly:

> The rails and their ties are now excluded from the explore walk, the stats bar and the legend — they are layout physics, not content. Focus on selection now actually empties the canvas down to the neighbourhood

The release note adds the number: seven nodes of ninety-eight.

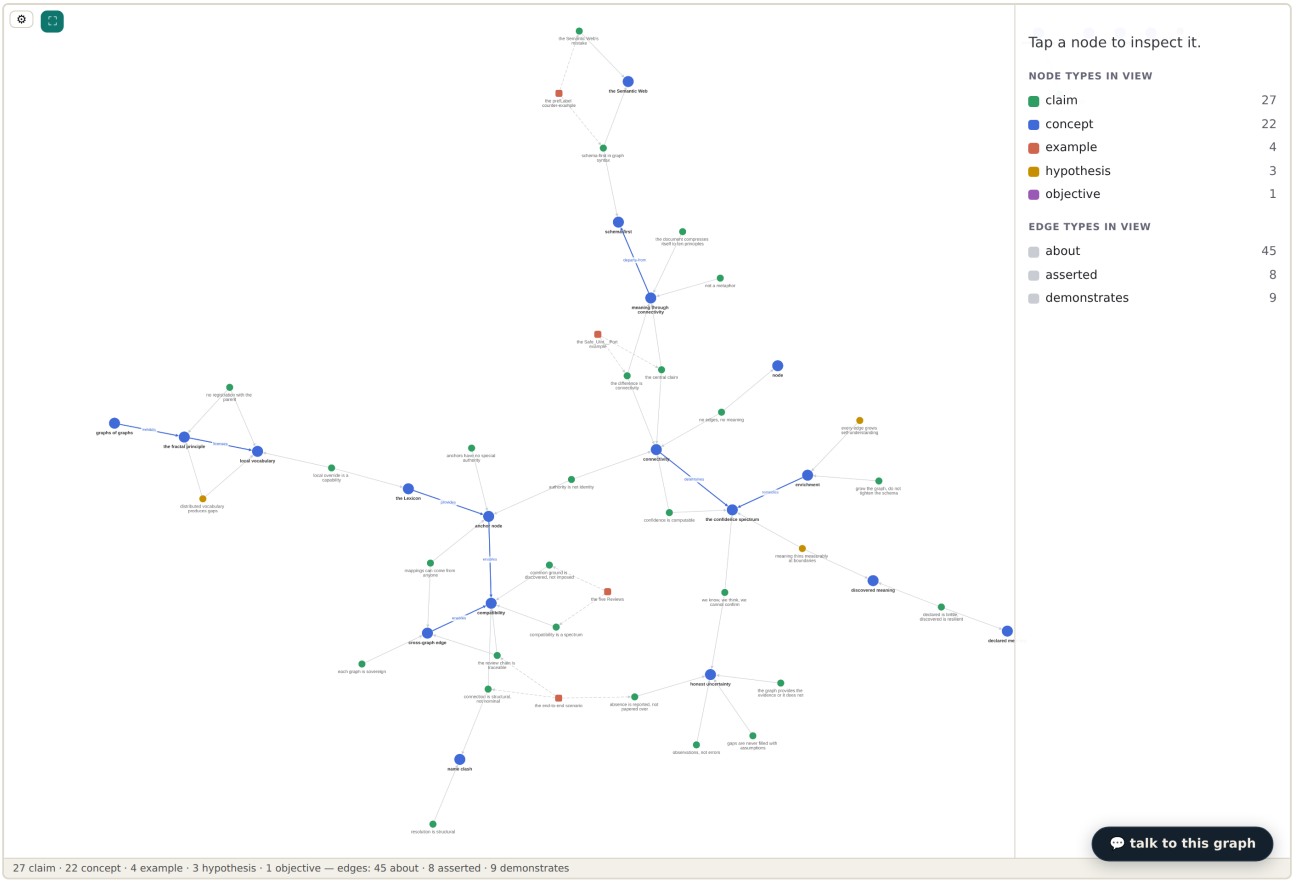

Figure 15. `/v2/universe/thinking-in-graphs.html#graph` at tag `v0.4.31`, 25 August 2026. The `#graph` fragment itself is one of the six fixes: at moment 2 the founder had scrolled to a static drawing and mistaken it for the live view.

The inversion: pages that broadcast their own state

The founder's follow-up message, sent in chat the same day and recorded verbatim in the same brief, is the best single idea in this chapter:

> great, so how can we add some more info that is useful to you? (since the prob here is that the recorded doesn't have programatic access to the page it is being recorded
>
> what if we add a debug pane (bottom right) that has that info you asked for?

The recorder cannot see inside the page. It only captures pixels. So instead of giving the recorder access, put the state into the pixels.

Eight minutes later, v0.4.32:

> The state pane: the page broadcasts its state into the pixels a recording captures. […] Bottom right, off by default (it exists for narrations and debug sessions), enabled by
>
> # debug on the URL or the reader options popover: the site version, document and hash, a
>
> running clock that joins each screenshot to the recording's own timeline, the current selection, layout and look, the sources that are on, the explore state and hidden edge kinds, the visible counts, and the last action taken ("last: gpeaks→1", "select connectivity"), the line that disambiguates "this" and "here" in a narration.

Two details make this more than a debug overlay.

The clock. Every screenshot carries a running clock, which means every screenshot can be placed on the recording's timeline independently of whatever metadata the recorder produced. If the two disagree, the pixels win.

The last action. "last: gpeaks→1". When a narrator says "I just did this and look what happened", the pane says what "this" was. That is the single line that converts an ambiguous sentence into a reproducible one.

The release note also says who the text is written for: "Short high-contrast monospace lines, written for OCR and vision models as much as for eyes."

And it closes the loop back to programmatic access rather than replacing it:

> The same truth is published as window.__uniState(), so a tool that does gain programmatic access reads exactly what the pixels say.

One source of truth, two readers. The pane and the API cannot disagree, because the pane renders the API.

The rule for anyone building anything an agent must diagnose

The retrospective states it as a transferable learning:

> **Screenshots need state in the pixels.** The narrated-review round exposed that a recording cannot see page state; the state pane put version, selection, sources and the last action into every screenshot. Any surface an agent must diagnose from images should broadcast its own state.

If you take one thing from this chapter into your own project: **put a small, switchable panel on every interactive page that prints the version, the current selection, and the last action taken.** It costs an afternoon. It pays for itself the first time somebody sends you a screenshot with a complaint attached.

Why this is easier than writing a specification

The narrated review is faster than written feedback, but the deeper reason it works is that it changes what kind of feedback is possible.

A written bug report is a claim about a system. A narrated review is a recording of a person's experience of a system, in order, including the parts where they were confused, tried something, and it turned out to be working after all. Moment 6 above contains "It doesn't work. Actually, it's working now." A written report would never contain that sentence, and the sentence is diagnostic: something took long enough to respond that a person concluded it was broken.

The brief's own summary of the trade:

> Ten moments took three and a half minutes to record and carried six actionable findings, two of which (the explore pollution, the board coverage) a written note would likely have described less precisely than one screenshot each did.

Where the live estate shows this. The narrated review is published at `/v2/memos/27-founder-review-narrated-viewer-walkthrough.html`, moment by moment, with the findings table and the uncertain corrections. The second narrated review, over the WCLM, is at `/v2/memos/33-founder-review-the-detective-play-book.html`. The state pane is still on every reader page: add `#debug` to the URL.

9 · The experiments

After this chapter you will know how a "crazy experiment" travelled from a voice memo to a tested, folder-structured subsystem in eight releases across one afternoon, and you will have a template for running experiments in your own project without them either dying quietly or taking the project over.

Building a lab

Every project of this kind acquires a graveyard of half-finished ideas. This one has a different arrangement, and the arrangement is stated in the memo that started the biggest experiment of the six days.

Brief 31, 26 August, the founder's voice memo:

> Okay, so using the artefacts that we've created, but I think you probably want to do this in a separate folder because this is in a sort of bit of a crazy experiment. But I would like to see how it can actually work.

And a little later, twice more:

> So what I would like to kind of create is a mini engine again. Do this in separate code, so that we can then figure out how to bring this back.

Three constraints in one breath: **separate folder, separate code, separate page**. The release note for v0.5.2 records them as instructions and takes them literally: the whole thing was built at `/v2/wclm/` with its own code and its own page.

This is the pattern. An experiment gets its own address so that it can be abandoned without surgery, or promoted deliberately. The founder says the second half out loud ("figure out how to bring this back") at the moment he asks for the first half. The experiment is scoped and its exit is named before a line of it exists.

Six releases later, at v0.5.9, the founder wrote the frame down completely:

> in fact view the development of these minor data/operators visualizations as a way to PoC and experiment with all sorts of ways to see, run, visualise and debug these operators (in a smaller problem space and code base), in a way that we can then apply the best ones (ie the ones that worked) to the main WCLM UI and workflows

A small problem space is a lab. Try things there where the blast radius is small, and promote the ones that work into the main surface. The v0.5.9 release note names which candidates were already promoted and which are queued.

What the experiment was

The idea in brief 31 is a good one to explain because it is not really about language models at all.

The founder's argument: a language model is not picking the next word at random. It has built a rich internal model of the world and consults it. So, he asks, what if we build something with the *shape* of a transformer, over graphs we already have, where nothing is learned and everything is named?

> the main thing I want to map and visualise is the look back and the attention, and that loops that were created between the different layers. So I think would be cool is why don't we maybe start with five layers or seven layers, you know, like neural networks

And the query flips. Not "what word comes next" but:

> instead of maybe that's a better analogy, but this is not about about creating just embeddings. It's about visually representing and give us the example and the provenance and the paths between a component, an example, another example, and the concept. And I think this is where maybe instead of going from predicting the next word, we can be what is the definition of something?

He also named it. Not a large language model but, in his words, "a words content language model": the WCLM.

What shipped, in one afternoon

v0.5.2, 13:44. Six layers, each a pure deterministic function: tokenise, resolve, attend, bind, expand, converge. Tokens are content hashes, so the same word tokenises identically in every document with no registry anywhere. Every weight is a stated formula written into a world file rather than a fitted number, which means, as the release note puts it, "training the model is editing graph inputs and meaning packs, never fitting numbers". The page draws the six layers as columns with weighted arcs between them.

One detail from that note is the best small story in the release table:

> one ranking bug in this very build was fixed by editing the bind formula, the training loop working as designed

The engine ranked something wrongly. The fix was to edit a formula in a data file. That is what "nothing is learned and everything is named" buys you: a bug in the model's judgement is a text edit.

v0.5.3, 14:17. Thirty-three minutes later, every box on the page is clickable and explains itself in both directions: what produced it, with the reason on each wire, and what it feeds. Appendix A reproduces the memo that asked for this and annotates it segment by segment.

v0.5.4, 15:35. The detective playbook. Strict layer adjacency after the founder caught a wire jumping (Chapter 7), negation handling after he ran "meaning without connectivity" and got the wrong answer, a normalise stage that repairs misspellings and says what it repaired, and the pipeline as eight reusable blocks you can toggle off and drag to reorder.

v0.5.5, 16:10. Three separate instructions that arrived across one afternoon, shipped together. Words get multiple senses across industries, with the document's own sense first; picking a different sense for "graph" makes the engine say which of the universe's claims stop applying. Layers can hold several engines side by side. And every engine declares its input and output types, so compatibility becomes structural: an engine placed where its input type has not been written is skipped, with the type named.

v0.5.6, 16:27. Analogies. The memo's ask, in the founder's words: to explain this material to somebody from finance, do not repeat the words, find the equivalent concept in their world. "Graphs of graphs" becomes spreadsheets of spreadsheets. An anchor node becomes the chart of accounts. Sixteen concepts mapped into three audiences, each carrying the reason for the mapping, and an honest "no analogy authored yet" wherever the register has a gap.

v0.5.7, 17:20. The restructure, which gets its own section below.

v0.5.8, 18:08. An iPad review of v0.5.7 with four screenshots and four findings, each fixed where it pointed.

v0.5.9, 18:47. The code itself gets the graph treatment.

Eight releases. Five hours and four minutes from the first to the last.

The restructure: twelve folders

Brief 36 is a typed message, not a voice memo, and it is short. It points at a file explorer that had been built for the document five releases earlier and asks for the same treatment for the engine's own building blocks:

> Great, now I would like to work and fine tune each of those operators individually
>
> Like you did for the document's folder (file explorer + views)

Then a list: put them in a dedicated folder, create a view of the files, create the markdown and html and js inside, schemas as files, examples and sample data, a reusable workbench for execute and test and debug, and visual representations of the architecture.

What shipped at v0.5.7 is a real restructure, not a presentational one. The twelve operators moved into twelve folders, and the file in each folder is the code the engine actually imports. The shared engine file went from 374 lines to 161. The diff is 159 files changed and 12,705 lines added. Seventy-four files live under the operators directory at that release, and eighty-six by v0.5.11.

Each folder holds six kinds of artefact, and the six are worth listing because they are a template:

- **the code** the engine really imports, not a copy;
- **the book page**, markdown, explaining what it does and walking through the transformation with an ascii architecture diagram;
- **the schema**, generated from the code's own declaration and gate-checked against it, so the description of the contract cannot drift from the contract;
- **the official data**, with the provenance of each item as a field rather than a comment: standard across every document, authored for review, or derived by another transformation;
- **the example vectors**, real input and output slices captured by running the engine. Thirty-nine across the twelve, and every one is replayed by a gate on every build;
- **the workbench**, a reusable page shell with execute, test and debug, which each operator can extend with its own controls.

The model for it had shipped five releases earlier, at v0.5.1, for the pilot document: a file tree on the left, the file on the right, every file readable as exact bytes or as a data-driven view, every file deep-linkable.

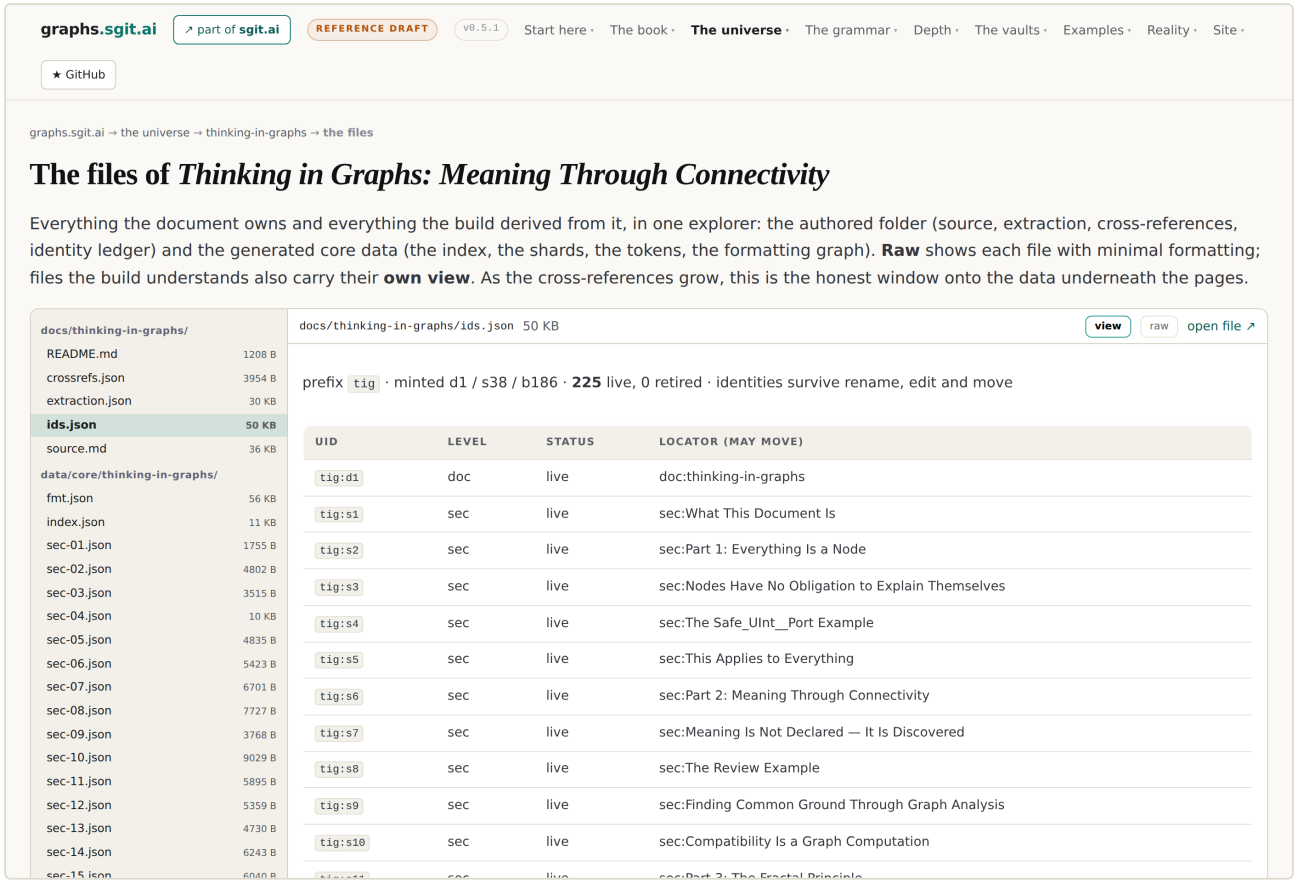

Figure 16. `/v2/universe/thinking-in-graphs.files.html` at tag `v0.5.1`, 26 August 2026, deep-linked to the pilot document's identity ledger.

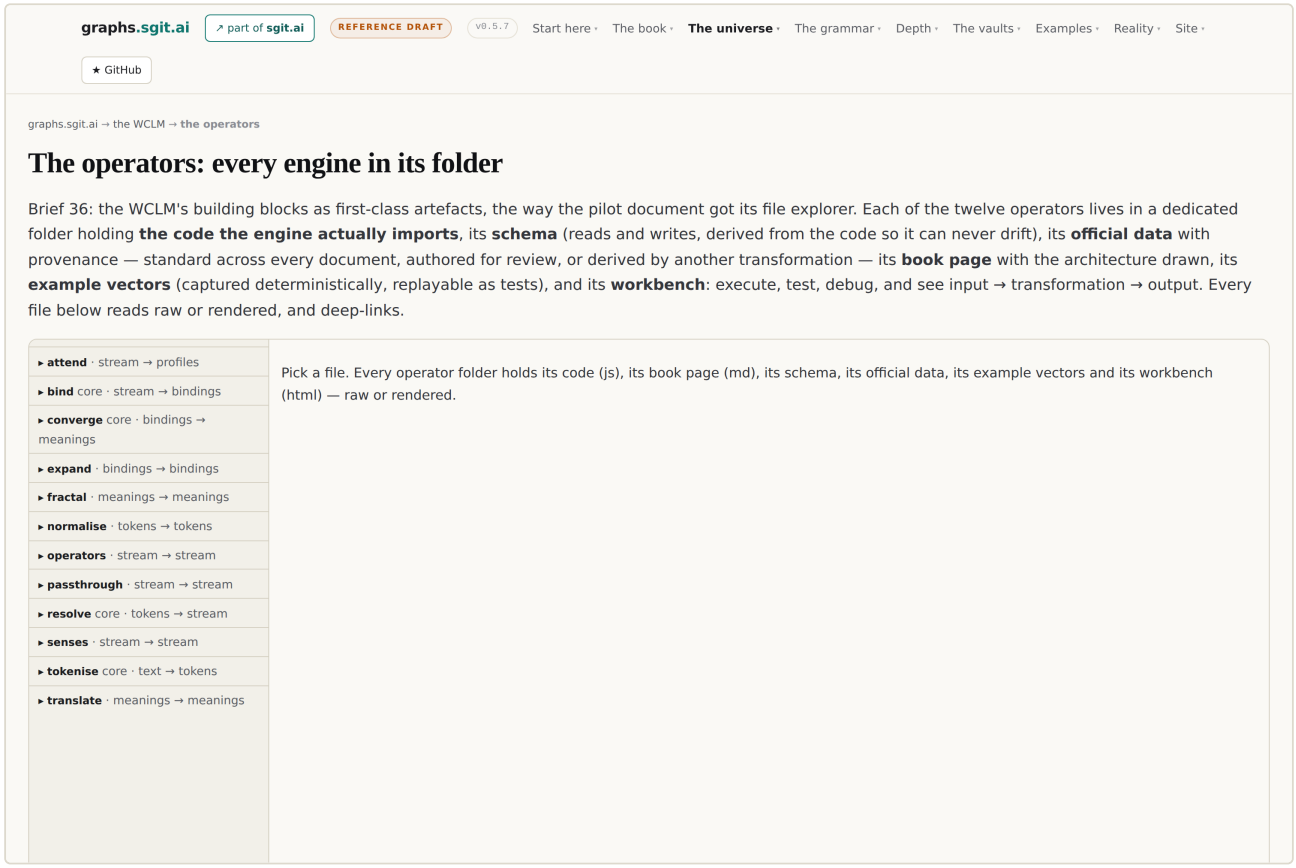

Figure 17. `/v2/wclm/operators/` at tag `v0.5.7`, 26 August 2026.

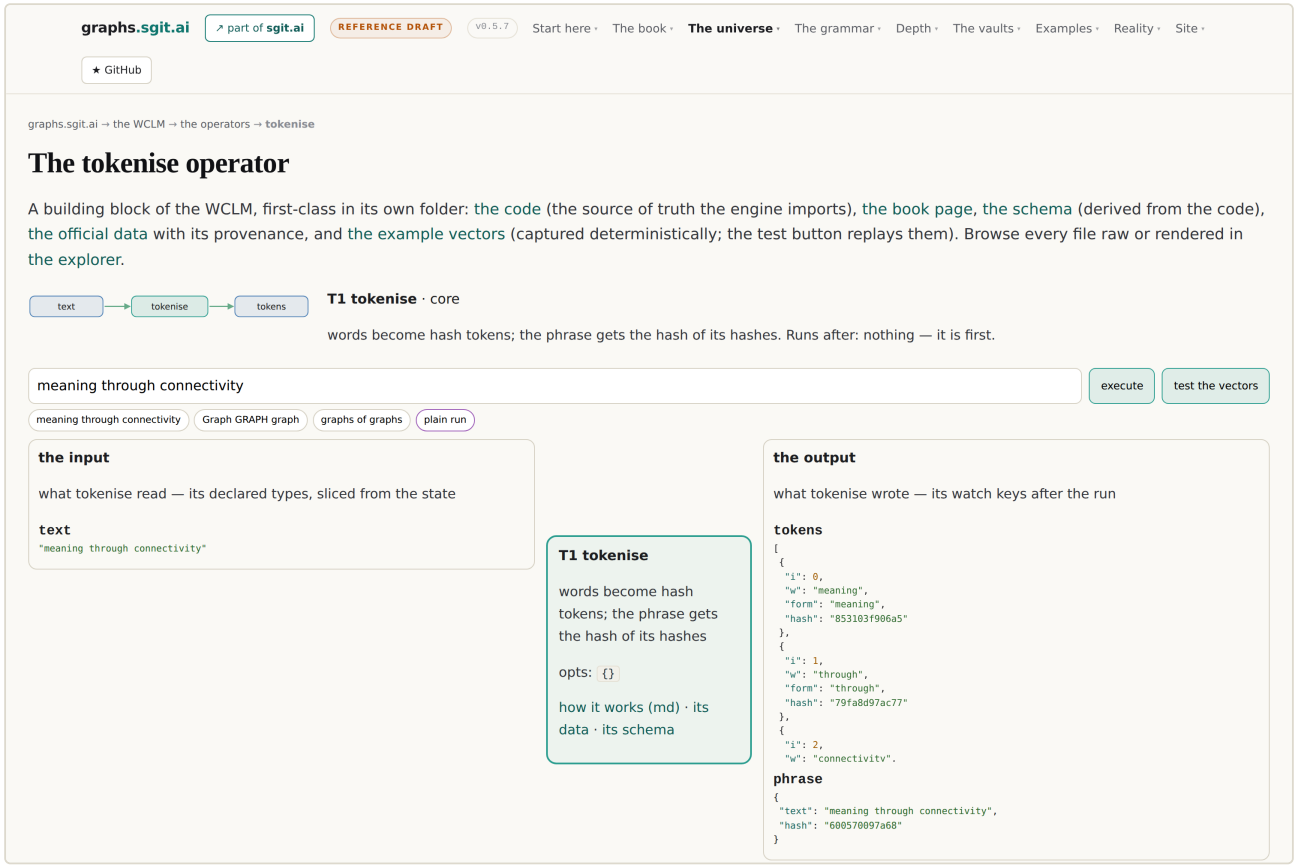

Figure 18. `/v2/wclm/operators/tokenise/` at tag `v0.5.7`, 26 August 2026.

Two things about this restructure are worth an author's attention even if they never build an engine.

The schema is generated from the code and gate-checked against it. A description of what something does, that cannot disagree with what it does, because a build fails if they diverge. Apply that idea to a book: a summary generated from the chapter, with a gate that fails if the chapter changes and the summary does not.

Every example is replayed on every build. Thirty-nine captured input-output pairs that have to keep producing the same output forever. The release note calls them "deterministic by construction", because they were captured by running the real engine rather than being written by hand.

Keeping zooming

Brief 37 arrived minutes after v0.5.8 went live, and it is four bullet points long. The first one:

> on the JavaScript pages, can you add a couple views that explain in a graph and visual way what is going on: I know JS very well, but at the moment since I don't have the context you have, those scripts (although small) are still hard to read and understand what is going on. Basically can you apply the graphs of graphs approach here […] Basically think : what would Brett victor do to explain and visualise what this code is going

The ask is: apply the project's own extraction discipline one zoom further in, to the source code itself.

What shipped is the same discipline exactly. Each operator's code is sliced into contiguous authored segments. Each segment is a node with a kind, a description written for somebody who already knows JavaScript, the variables it uses and their roles, what it reads and writes, and edges to the segments it drives. Each segment is

anchored to the exact text of its first line, the build resolves those to line ranges, and the ranges must tile the file completely. A gate fails the release the moment the code and its description drift apart.

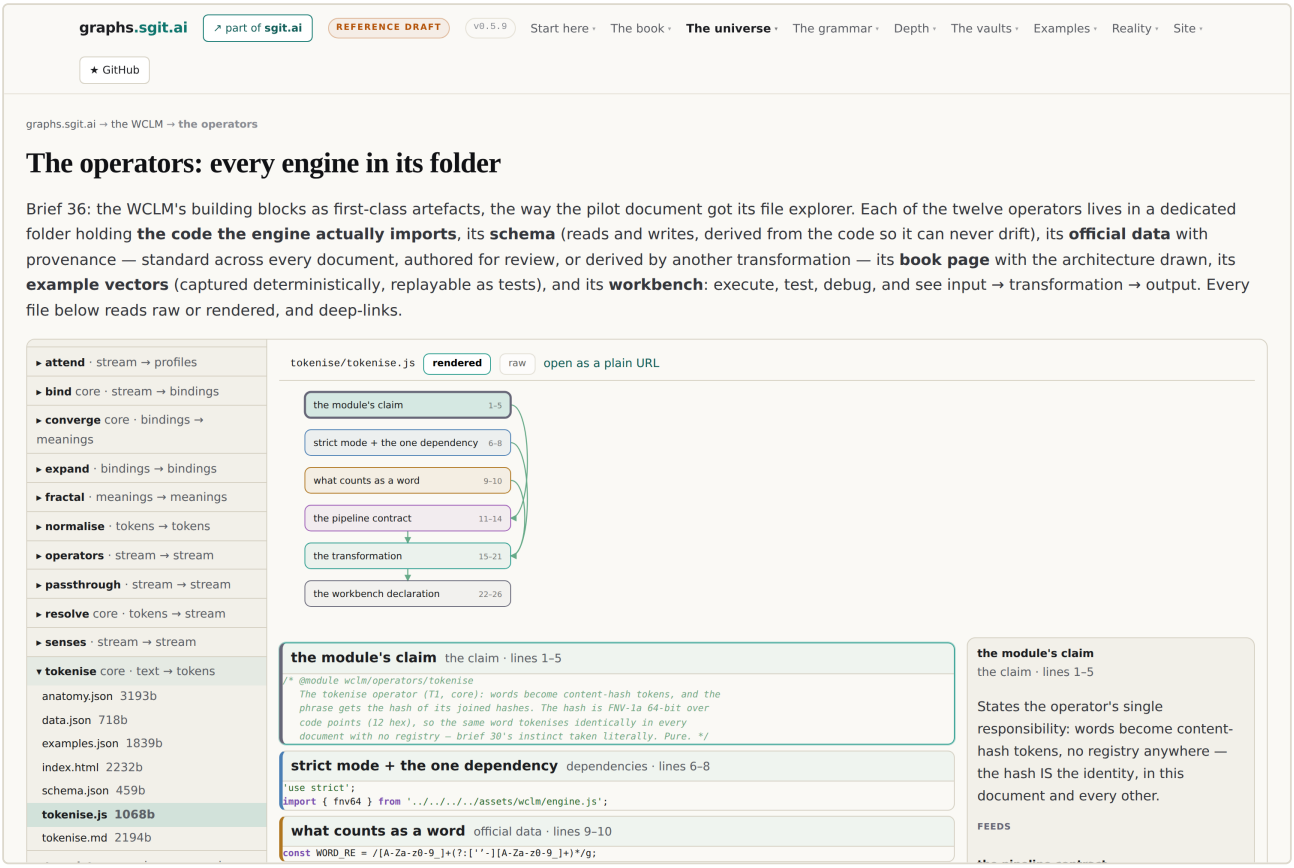

Figure 19. `/v2/wclm/operators/index.html#tokenise/tokenise.js` at tag `v0.5.9`, 26 August 2026.

Click a box in the flow, a block of code, or an entry in the explanation, and the same segment lights in all three.

Notice that the mechanism is the one from v0.4.5, unchanged: anchor an assertion to the exact bytes it is about, and let a build gate check the anchor. It was applied to a document first, then to the document's own structure, then to an engine's data, and now to source code. That reuse is not a coincidence, it is the fractal claim the whole book is about, applied by the people making the argument to their own tools.

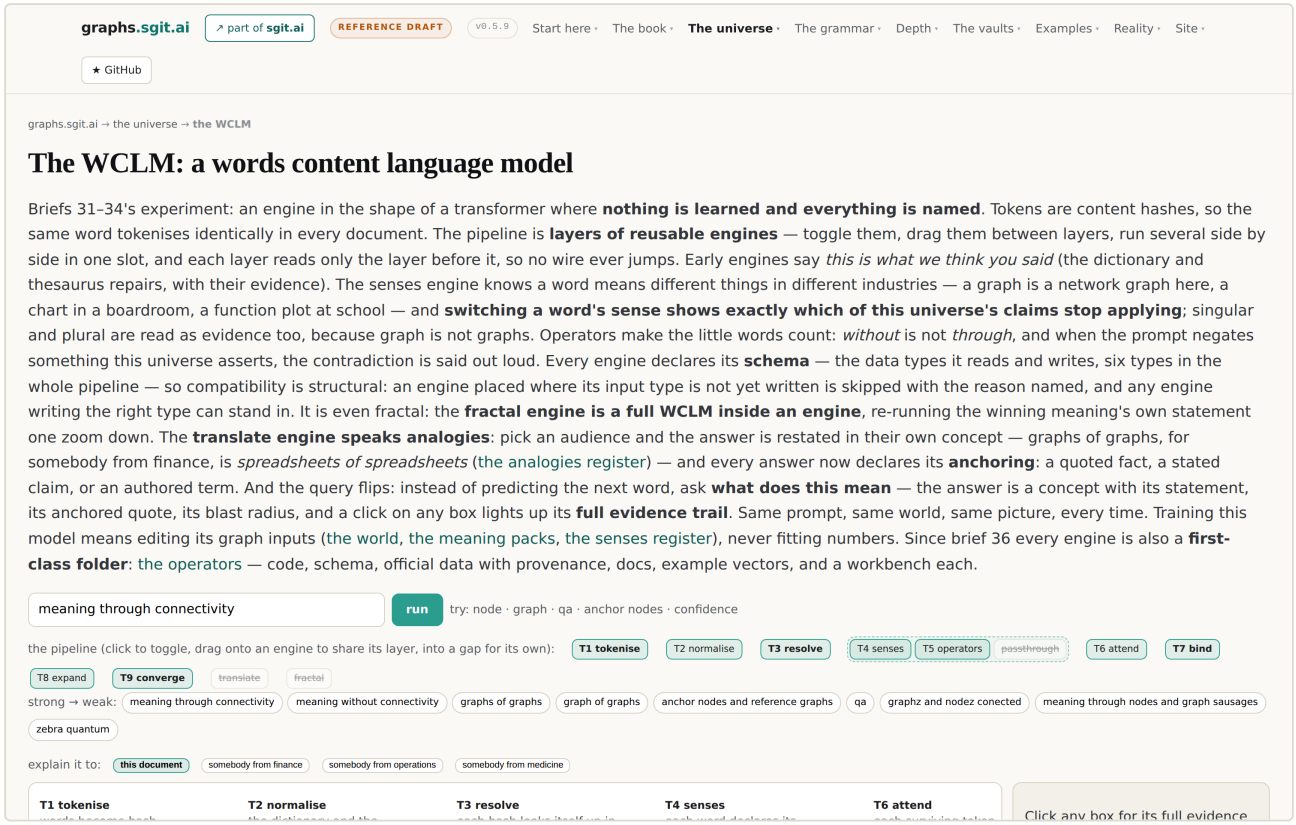

Figure 20. `/v2/wclm/` at tag `v0.5.9`, 26 August 2026.

How to run experiments this way

Four rules, each of which the corpus states in its own words:

1. **Separate folder, separate code, separate page.** So it can be abandoned without surgery.
2. **Name the exit at the start.** “So that we can then figure out how to bring this back.”
3. **Register it honestly.** The methods register carries thirty-five techniques; the WCLM is entry thirty-five and its status is `experiment`, not `in use`. Every other entry names the release where it first shipped.
4. **Promote what worked, and say which.** The v0.5.9 note names the promotion candidates by name: the chip-and-wire execute view (which already shares its code with the main page by construction), the anatomy view’s one-identifier-three-views selection, and the in-page replay.

An experiment nobody promotes is a hobby. An experiment nobody scopes is a rewrite. This one was both scoped and promoted, in public, in nine hours.

Where the live estate shows this. The WCLM is at `/v2/wclm/`, the twelve operators at `/v2/wclm/operators/`, and each operator’s workbench at `/v2/wclm/operators/<name>/`. The memos behind them are briefs 31 to 37 at `/v2/memos/`. The methods register, with the WCLM marked as an experiment, is at `/v2/methods/`.

10 · The founder’s craft

After this chapter you will know what to say to an agent, in what shape, and you will have five habits from a person who steered eleven build rounds in six days without writing a line of code.

This chapter is the one that transfers most directly, because it is about the half of the loop a person does.

Nineteen briefs and eighty-eight release notes are a large sample of one person instructing an agent. Read as a set, they have a consistent style, and the style is learnable. Five habits do most of the work.

---

1. Point at a thing, then say what is wrong with it

Almost every instruction in this corpus is attached to a specific object on a specific screen.

> when I open up the the tignogen graphs, the graph it doesn’t fit. It’s not in a fit state. So we should literally you should run the fit here
>
> the graph options gear was only visible after scrolling to the very top
>
> The screenshot shows the wire running from L2 resolve straight to L4 bind

Compare that with the kind of instruction that produces nothing useful: “the graph view needs work”, “make the navigation better”. The corpus contains essentially none of these, and the difference is not politeness or precision of language. It is that a located complaint contains its own test. An agent can tell whether it has fixed “the graph opens unfitted”. It cannot tell whether it has fixed “needs work”.

The strongest form is a screenshot plus a sentence, which is exactly what the narrated review of Chapter 8 industrialised: ten screenshots, three and a half minutes of talking, six findings.

2. Say the principle, not only the fix

The most valuable memos in this corpus contain a rule alongside the request. Brief 26, about fixed nodes, produced the line that the retrospective calls the era’s deepest design law:

> Every node move costs the viewer their mental picture

That sentence is not an instruction. It is a principle from which a dozen instructions can be derived, and the agent derived four mechanisms from it across four releases: stable add, never moving the viewport uninvited, pinned summits, alignment rails.

The same shape appears elsewhere. Brief 22’s instruction 4, in the founder’s own words:

> I want you to start thinking in terms of sources of nodes to add. So it’s almost like not necessarily a view, but sources that we adding to the thing

That is a reframing, not a feature request. “Views” and “sources” imply completely different architectures: views are a fixed list somebody maintains, sources compose. The whole node-pack design comes from that one sentence.

An agent will implement a feature request accurately and narrowly. Give it the principle and it will apply it in the places you did not think of.

3. Name the reference

Twice in this period the founder handed the agent a name instead of a specification, and both times it produced more than a specification would have.

At v0.4.35 he named Bret Victor’s talk *Inventing on Principle* as the experience the viewer was aiming at. The agent went and looked, found that half the viewer’s strongest features were already unwitting implementations of Victor’s patterns, and wrote a standing register mapping the patterns to the releases, with two fully specified gaps any future agent can pick up.

At v0.5.9, in brief 37, he wrote:

> Basically think : what would Brett victor do to explain and visualise what this code is going (to an audience that knows what JS is, so no need to explain what a variable is)

Two useful things in one bullet: a reference for the standard, and a statement of who the audience is not. “No need to explain what a variable is” removes an entire register of output the agent would otherwise have produced.

Naming a reference is high-bandwidth. It transfers taste, which is the thing hardest to write down.

4. Ask “does this make sense, any questions?”

The corpus is full of the agent’s questions, because the founder keeps asking for them. Every brief in `v2/briefs/` ends with a numbered list of the questions the agent owes answers on, and that section exists because it is asked for.

This matters more than it sounds. An agent that is never asked for questions will not volunteer confusion; it will resolve ambiguity silently and plausibly, and you will find out which way it resolved when you see the result. An agent that knows it will be asked writes the ambiguity down as it goes.

The founder’s answering pattern is also worth copying: he answers in batches, on his own schedule, and each answer ships. Brief 26’s four questions came back answered and all four shipped in one release, v0.4.29, whose note opens “The founder’s four answers, applied”. Brief 38’s two held questions were answered the same evening and reshaped an entire handoff.

The queue works because the questions are written down in a file with a URL, rather than being asked in a conversation that scrolls away.

5. Praise the specific thing, then correct

The corpus’s verdicts are recorded before the corrections, deliberately. Brief 33’s header carries the line “Verdict recorded first: ‘this is looking really good.’” Brief 32’s carries “this WORKED really WELL … you did a great job at those first transformations, which now we can add more and tweak.”

The praise is not decoration and it is not, in any meaningful sense, for the agent’s feelings. It is information. “You did a great job at those first transformations, which now we can add more and tweak” tells the agent that the transformation layer is settled and the work is now additive. Brief 20’s cancellation of an entire plan opens with “the brief is amazing, right? Like it’s really cool”, and then names the six of ten files that have to change. The praise is the boundary of the correction.

An agent given only corrections will over-rotate, because it cannot tell which parts of its work survived.

What he did not do

Five absences from this corpus are as instructive as the habits.

He did not write code. The record contains no instance of the founder editing a source file. The two places where he made a technical decision, moving rather than copying the first edition at v0.4.0, and deleting the 108 redirect stubs at v0.4.7, are structural calls, not implementations.

He did not write specifications. There is not one document in the second edition’s corpus written by the founder in advance of a build that specifies behaviour. The only planning document in the whole period, the ten-file dev pack at v0.3.27, was written by the agent and largely cancelled by the founder’s response to it.

He did not review code. He reviewed running software, on his own devices, with real data. The iPad round of v0.4.26 found two bugs no code review would have surfaced.

He did not accumulate feedback. Twenty releases on 24 August means feedback arriving roughly every thirty minutes. The v0.4.9 note begins “Founder feedback on using v0.4.8, all of it acted on”, twenty-nine minutes after v0.4.8. Batching feedback into a weekly review would have made every one of those rounds worse, because the agent would have been three days downstream of the thing being reviewed.

He did not mediate between the two agents. He commissioned the exchange in Chapter 6 and let them write to each other.

The trade this represents

Put the five habits together and the founder’s job is: decide what matters, look at the real thing often, say precisely what is wrong with a specific object on the screen, supply the principle behind the fix, and answer the questions that come back.

That is a substantial job and it is not a passive one. Two of the six days ran past fourteen hours. But it is a different job from the one an author normally does, and the difference is worth naming: **almost none of it is production, and almost all of it is judgement.**

The parts of writing a book that a person is uniquely good at, deciding what the book is for, noticing when something is subtly wrong, knowing which of two right answers is the one you meant, are exactly the parts this arrangement leaves with the person. The parts that are laborious are the parts it moves.

That is the case for the method, stated as strongly as the evidence allows. Chapter 12 states the case against.

Where the live estate shows this. All nineteen memos are at `/v2/memos/`, each with its instruction table and its questions. The immediate-connection register, which is what came of naming Bret Victor, is at `/v2/dev-pack/design-00-the-victor-register.html`.

11 · The playbook

This chapter stands alone. It is written so it can be torn out, printed, or pinned up without the rest of the book around it. Everything in it is drawn from the six days described in Chapters 1 to 10, and every claim has a chapter behind it.

Before you start: what this is and is not

This is a way of writing a book, or any large structured document, in which an AI agent does the production and you do the judgement. It assumes:

- you are willing to run a git repository, or to let the agent run one for you;
- you can review a working artefact rather than a draft;
- you want the process to leave a record you can check afterwards.

It does not assume you can program. It does not require a graph database, or any database. The system this playbook is drawn from stores everything as markdown files and JSON files in a git repository.

Day 0: set up five things

Set these up before you write anything. In the project this book describes, the pipeline shipped as release v0.1.0 before there was a single chapter, and that ordering is the reason the next eighty-seven releases were possible.

1. A repository with one release branch

One branch that is the published one. Every change lands there. No long-lived feature branches: in six days and ninety-seven commits, this project had three merges total.

2. Automatic validate, tag and deploy

A push to the release branch should run your checks, and only if they pass, tag the commit and publish. No manual release step. The consequence, worth having, is that every version of your book that ever existed is recoverable by name forever. Every figure in this book was re-taken from a tag.

3. Seven checks to start with

Do not design a test strategy. Copy these, adapt the ones that apply:

1. **Version agreement.** One file owns the version number; everything that displays it must match.
2. **Internal links.** Every link and image path resolves to a file that exists.
3. **Canonical addresses.** Every page declares one canonical URL and it is right.
4. **Structural completeness.** Every section of your work is listed in whatever index you maintain, in both directions.

5. **A vocabulary tripwire.** Pick the one word or construction your project has banned and fail the build if it appears. This project bans a vague relationship name, `relates-to`, and permits it only where a page is quoting the ban.
6. **A secret tripwire.** Nothing in the tree may look like a key or a token.
7. **A structural balance check.** Something cheap that catches the class of error your format allows silently. This project counts opening and closing `<div>` tags, after four pages shipped with a note element closed by the wrong tag and browsers accepted it silently.

Add the eighth check the first time something goes wrong. That is how this project got from seven to sixteen.

4. A briefs folder

`briefs/`, numbered files, one per instruction round. See Day 1.

5. A release table

A page listing every release: version, date, and a paragraph on what changed. Prose, not commit messages. This will become the most useful artefact you own. Six days later it is the only reliable account of what happened, because unlike a commit log it was written by somebody who knew why.

Day 1: your first memo

Talk, do not type. The nineteen briefs behind this book are mostly transcribed voice memos, and they are richer than typed instructions because people say more when they speak.

Record while using the thing, not while thinking about it. Every memo in this corpus was recorded with the deployed site open.

What to say, in this order:

1. **The verdict, first.** “This worked really well” or “this is not right”. Say which part is settled. It tells the agent whether the next work is additive or corrective.
2. **The specific complaints, each attached to an object.** “When I open X, Y does not happen.” Point at things. If you can send a screenshot, send one.
3. **The principle behind the complaint, if you have one.** “Every node move costs the viewer their mental picture” was worth four releases. A principle is worth ten feature requests.
4. **The direction, marked as direction.** Say out loud the things you are thinking about but do not want built yet. Ask for them to be recorded, not built.
5. **The question.** “Does this make sense, any questions?” Ask every time.

What to do with the recording:

Have the agent write a file, `briefs/NN__whatever.md`, with three parts in this order:

- the transcript, verbatim, including the transcription errors, never edited afterwards;

- the agent's reading: a numbered table of every instruction and, in the second column, **what it commits the work to** (not what to do: what is now in scope and, by omission, what is not), under a heading that says whose reading it is;
- the questions the agent could not answer, and the defaults it took anyway, each with the way to reverse it.

Then reference the brief number in the release note when the work ships.

Ten minutes per round. It is the highest-return habit in this book.

The loop, in five steps

Run this as often as you can stand.

1. **Record a memo while using the current version.**
2. **The agent publishes it verbatim, with its reading and its questions.**
3. **The agent builds, and writes the release note as part of the build.**
4. **The push runs the gates; if they pass, a machine tags and deploys.**
5. **You open the deployed thing and start again at 1.**

Target: one round in under an hour. This project's median gap between releases was 31 minutes over six days. You will not do that, and you do not need to. What matters is that a round is short enough that you review the real thing rather than a memory of it.

Rules that make it work

Never edit a transcript. Add to the file. Do not rewrite it. The transcript is the only independent record of what you asked for, and if the agent can edit it, it is not independent.

Every bug you find becomes a test before the release ships. When the founder found two bugs on an iPad, that release added seven checks that would have caught them. You should not have to find the same class of thing twice.

Corrections get a release, not a silent edit. When something in your published work is wrong, fix it in a release with a row in the table saying what was wrong. The project's own rule, from its second day.

Record debt in public. One component in this project grew from 202 lines to 434 against a stated budget of 250. Every release that grew it said so, and the remedy was named. Debt you have written down behaves completely differently from debt you have not.

State the third category. Built, not-mentioned, and *recorded as direction*. Without the third one, every idea you say out loud is either work or waste, and you will stop thinking out loud.

Give experiments their own folder. And name the exit at the start: "do this in separate code, so that we can then figure out how to bring this back."

Put the state in the pixels. Every interactive page should have a switchable panel printing the version, the current selection and the last action taken. It costs an afternoon and it makes every screenshot you ever send diagnosable.

What to expect to go wrong

From Chapter 7, in the order you will meet them.

Your agent's tests will pass and the thing will still be broken. Automated verification tests what somebody thought to test. Use the real thing, on a real device, with real data, regularly. Two of this project's best bug reports came from an iPad session, and one from scrolling a page sideways.

Your verification tools will lie to you. A stale process holding a port cost this project an hour of debugging code that was already correct, deterministically. If a stubborn failure is fixed by killing a process, that is a finding about your harness. Never reuse a port between runs, and always kill what you spawned.

Your agent will write clean generalisations that more data breaks. The remedy is to compute your claims on every build so the next batch of evidence breaks them out loud instead of quietly.

Your agent will build the thing you were musing about. Hence the third category.

Two agents will collide. Give each one a tree of files it owns and forbid it from touching another's; have every agent fetch the release branch and tags before choosing a version number; and when one has comments about another's work, have it write a document addressed to a peer rather than silently refactoring.

Signs the loop has stalled

- **The gap between releases is growing.** Something has become expensive. Usually it is a missing gate: the agent is hand-checking things.
- **Your memos are getting longer and less specific.** You have stopped using the thing and started theorising about it.
- **The agent has stopped asking questions.** It is guessing. Ask for the questions explicitly, every round.
- **Release notes have become one line.** Nobody will be able to reconstruct this week.
- **You are reviewing plans rather than software.** The plan is not the product, and a plan reviewed is a plan that has consumed a round without producing evidence.

The honest costs

Read Chapter 12 for the full version. In brief:

- **This is not passive.** Two of the six days behind this book ran past fourteen hours of releases. The method does not require that pace, but it does require your attention every round, and the attention is judgement, which is tiring in a different way from production.
- **You will ship things you have not fully read.** That is what the gates are for, and it is still true.

- **You will produce more infrastructure than book.** In six days this project produced an entire working surface for writing its second edition and none of its chapters. That was a deliberate choice, taken on day three, and it was the right one, but if what you want is chapters by Friday this is the wrong method.
- **The record is public and permanent.** Everything in this playbook assumes you are willing to publish your failures on the day they happen. If you are not, most of the benefits go away, because the value of the record comes from it being honest.

The one-page version

1. Pipeline before prose: repository, gates, automatic tag and deploy, on day zero.
2. Talk while using the thing. Point at objects. Give the principle.
3. Transcript verbatim, then the agent’s reading, then the questions. Never edit the transcript.
4. Ship small and often. Every release gets a paragraph saying what changed and what was verified.
5. Every bug a human finds becomes a test that day.
6. Record debt, corrections, and directions-not-built in public.
7. Experiments get their own folder and a named exit.
8. Suspect your harness as much as your code.
9. Ask “any questions?” every round, and answer them in batches.
10. Keep the tags. They are how you check, later, that any of this was true.

12 · What it costs, and where it loses

After this chapter you will know four situations in which the method in this book is the wrong one, what it charges that nobody mentions, and what the six days did not produce.

What was not produced

Start with the largest fact, because it is the one most likely to be glossed over.

In six days and eighty-eight releases, this project did not write any of the second book.

Not one chapter. The `/v2/` tree at v0.5.11 contains an extraction pipeline, a document reader, a core graph, an identity ledger, a deterministic transformer with twelve operators, a file explorer, nineteen briefs, thirty-five registered methods, a retrospective, and three empty book folders. It contains no prose of the book it exists to produce.

This is not a hidden failure. It is what the founder asked for on day three, in brief 20, in these words: build all the reference material first, decompose it, and only then draw the plot lines. The retrospective closes the v0.4 era saying the same thing in the agent's voice: "The surface is built; v0.5 is what gets made with it."

But it is a cost, and it is the cost you should weigh first. If your deadline is Friday and what you need is chapters, this method will hand you a very good machine and no chapters.

What was extracted

The second fact of the same shape. The universe names twenty-one source documents. One has been extracted.

The retrospective says so plainly, under a heading that reads "Observations, held loosely":

> One document is extracted; twenty wait. Every mechanism built this era (folders, gates, shards, ledgers, graph pages) was designed to fan out without redesign. v0.5 finds out.

At v0.5.11, v0.5 still has not found out. The fan-out has not happened. Everything in Chapter 5 and Chapter 9 is built on a single pilot document, and the claim that it generalises across twenty more is a design intention with good evidence behind it, not a demonstrated result.

Where the corpus is silent, this book says so, and here it is silent.

The costs nobody mentions

Attention, not time

The obvious cost is hours, and the hours were real: two of the six days ran past fourteen hours between first and last release, and one spanned twenty-two.

The less obvious cost is the shape of the attention. Every thirty minutes, something arrives that requires a judgement: is this right, is this what I meant, is this good enough to build on. There is no long stretch of unbroken making, because the making is being done elsewhere. Some people find this energising and some find it exhausting, and the difference is worth knowing about yourself before you commit six days to it.

The method does not require this pace. Nothing in the loop breaks if a round takes a day. But the loop is only as good as the review at the end of it, and the review is the part you cannot delegate.

You will ship things you have not read

At twenty releases a day, nobody reads every line. This project's answer is the gates: eighty-four unit tests, sixteen validator checks, byte-identical rebuild proofs, replayed example vectors. Those catch a great deal.

They do not catch everything, and the honest statement is that this method involves publishing work you have not personally verified line by line, on the strength of machine checks and the fact that you looked at the result. If that is unacceptable in your domain, this pace is unacceptable in your domain.

The record has to be public to be worth anything

Most of the value in this book comes from the record: the verbatim briefs, the honest release notes, the debt written down, the hour lost to a zombie browser reported the same day.

That record is only useful if it is honest, and it is only reliably honest if it is public. A private log of your own failures is a log you will edit. The line "the round also burned an hour on a lesson worth recording" was written into a public table by an agent who knew it would be read.

If you are not willing to publish the failures, you will get a much weaker version of this method, because the incentive that keeps the record straight will be missing.

Debt accumulates in the open, which is not the same as not accumulating

The component that draws the graph went from 202 lines to 434 against a stated budget of 250, over roughly two days, and was still at 434 at v0.5.11. Every release that grew it recorded the growth. The remedy was named at v0.4.13 and has not been carried out.

The retrospective's framing is "honest debt beats hidden debt", which is true. It is also true that the debt is still there. A public record of a problem is not a solution to it, and a method that makes it easy to record debt makes it easy to keep recording debt.

The version number is not a measure of anything

Eighty-eight releases sounds like a lot, and some of them are one button label. Release count measures the granularity of your release process, not the amount of work. It is a useful number for understanding the cadence and a misleading one for understanding output. The word counts, test counts and file counts in this book are more honest measures, and they are all in Appendix B and Appendix C, computed rather than recalled.

Four situations where you should do something else

The first edition of the book this project is writing has a page about where its own argument loses. The same courtesy applies here.

1. Your book has a fixed, known shape

If you already know the chapters, the order, and roughly what each says, the whole apparatus in this book is overhead. Build the universe first is advice for the case where you do not yet know what the book is. If you do know, write it: an agent will still help enormously with the production, and you need one branch, one gate on links, and a release table, not twelve operator folders.

The tell: if brief 20's complaint, "we are already defining the answer before we know what questions we're answering", does not describe a real risk for you, you are not in this book's situation.

2. You cannot publish the working record

Confidential material, an unfinished argument you are not ready to be seen holding, a publisher who wants an exclusive. If the transcripts and the failures have to stay private, most of the discipline this book describes stops being self-enforcing, and you should expect the record to drift towards flattery.

3. You are not going to look at the output every round

The loop's quality comes entirely from step 5. An agent shipping every thirty minutes to nobody produces a large quantity of confident, plausible work in a direction nobody checked. That is worse than slow work, because it is expensive to unpick and it looks finished.

If you can only review once a week, slow the loop to once a week. Do not let it run without you.

4. The work is mostly a single act of judgement

Some books are one argument, made once, that either lands or does not. Poetry. A short polemic. A memoir whose value is in the voice. The method in this book is built for work with a lot of structure and a lot of evidence, where the labour is in organising and checking rather than in composing. Applied to a book whose whole value is one person's sentences, it will produce a large amount of machinery around a thing that did not need any.

What it does buy

Against all of that, the honest positive case, stated in the terms this book has used throughout.

Six days of work with a complete, checkable record. Every claim in this book was verified against the repository. Every screenshot was re-taken from the tag its caption names. That is not normal. Most projects, six days later, cannot tell you why a decision was made.

A rate of correction that changes what you can attempt. When being wrong about a design question costs thirty minutes, you ask different questions. "Meaning without connectivity" was tested by running it, not by arguing about it. That is worth more than the speed itself.

Judgement concentrated where it belongs. The founder spent six days deciding things and none of it typing. The parts of the work that are uniquely human, knowing what the book is for, noticing something subtly wrong, choosing between two right answers, are exactly the parts the arrangement leaves with the person.

Infrastructure that compounds. Thirty-five techniques, each used in earnest before being written down, each naming the release where it first shipped, and the superseded ones still listed with their supersession recorded. That register is worth more on day thirty than on day six, and it exists because writing it down was part of the loop rather than a task for later.

The honest summary

This method produced, in six days: a working surface for writing a book, an unusually good record of how it was made, and no book.

Whether that is a success depends entirely on whether you agree with the decision taken on day three, which was to build the universe before finding the plot. The founder's argument for it is in brief 20 and it is a good argument. It is also a bet, and at v0.5.11 the bet has not yet paid out.

This book's own position, marked as the writing agent's judgement: the bet is sound for a book of this kind, where the material already exists and the difficulty is finding the story in it. It would be a bad bet for most books. And the parts of the method that are unambiguously worth stealing, the verbatim briefs, the gates, the honest release notes, the state pane, every-bug-becomes-a-test, are worth stealing regardless of which way you bet.

Where the live estate shows this. The first edition's own account of where its argument loses is at `/v1/about/participant.html`. The v0.4 retrospective, including the observations held loosely, is at `/v2/dev-pack/retro-00-the-v04-retrospective.html`. The three empty book folders are at `v2/books/`, each carrying its own commissioning prompt.

Appendix A · One brief, annotated

This appendix reproduces one complete brief and annotates it. Brief 32 was chosen because it is short enough to print whole, it is a voice memo rather than a typed note, and the release it produced shipped thirty-three minutes after the release it reviews. It is the loop of Chapter 1 in one document.

The memo is reproduced exactly as it is published at `v2/briefs/32__founder-memo__every-box-explains-itself.md`, broken into numbered segments for annotation. Nothing in the quoted text has been corrected. The annotations after each segment are this book's, not the original brief's.

The header the brief carries

> **Date:** 26 August 2026 **Source:** a voice memo recorded by the founder through the viewer's own loop ("voice memo via your ux"), sent with two screenshots of the WCLM: the default prompt, and the same prompt widened to "meaning through nodes and graph sausages" — which, as the founder noted with delight, nicely detects the word the universe does not have. **Status:** the founder's words are source material; the transcript is reproduced verbatim below. The instruction table and the questions are the agent's reading, marked as such. **Verdict recorded first:** "this WORKED really WELL … you did a great job at those first transformations, which now we can add more and tweak."

Annotation. Four things are fixed before a word of the memo appears: when it was said, how it was captured, whose words are whose, and what the verdict was. The last line exists because the agent is about to be given a list of changes, and without the verdict it cannot tell whether the changes are corrections or extensions. They are extensions.

Note also that the source line records the two screenshots and what they showed. The memo refers to them as "pic1" and "pic2" and would be unreadable without that sentence.

Segment 1 · The praise, and the thing that worked

> what also worked really well is the ability to put multiple works and phases where we go from the default one (pic1) into that with more words (pic2) which for example nicely detects works not used :) So for this next set of changes, can we experiment with this?

Annotation. "Works" is the transcriber's rendering of "words", twice. It is left in.

The substance is a report of a discovery: running the same engine on a longer phrase makes it visibly flag the words it has no use for. The founder found that by playing, not by reading a specification, and he is telling the agent that this behaviour, which was probably incidental, is now a feature worth building on.

This is the most under-rated kind of instruction. It is not "add X". It is "the thing you did by accident is the good thing; do more of that".

Segment 2 · The central ask

> First of all, every single box at every single layer needs to be clickable so we can zoom in into why that's there because that's kind of the point. Right? The point is to start to explain why we have certain words, why we have certain, you know, layers on our, you know, basically, Yeah. So every one of our layers.

Annotation. The instruction is one clause: every box clickable. The justification is the rest, and the justification is what makes the instruction actionable, because "clickable" on its own does not say what should happen on click.

"So we can zoom in into why that's there because that's kind of the point" tells the agent that clicking must produce an explanation of provenance, not a selection or a detail panel. Everything the agent built in the release afterwards follows from that clause rather than from the word "clickable".

The trailing "you know, basically, Yeah" is left in. It carries no information and it costs nothing to keep, and the moment you start deleting the parts that carry no information you have started editing the transcript.

Segment 3 · The principle, stated

> Right? So… and then let's start thinking about… I think you probably maybe below or maybe add to the right, or let's find a good place to put it. Like, the information about that layer. And it's it's the same thing as before. Think about it. It's it's graphs or graphs. Right? So every note, every item of every one of those layers now had the reason to be there, which is caused by downstream but also upstream. So we wanna start showing that.

Annotation. Two different kinds of content in one breath.

"Maybe below or maybe add to the right, or let's find a good place to put it" is an explicit delegation. The founder does not care where the pane goes and says so. The agent took the right-hand side on desktop and below on a phone, and recorded that as a default it chose, which is question 1 at the end of the brief.

"It's graphs of graphs" is the principle, and it is the reason this is a design instruction rather than a feature request. It says: the explanation of an item is itself a graph, with the item as a node, and it has edges in both directions. That is why what shipped has two lists, "because of" and "leads to", and why every entry in both lists is itself clickable. The founder did not ask for two lists. He said graphs of graphs, and two lists is what graphs of graphs means here.

"Caused by downstream but also upstream" is, strictly, back to front: an item is caused by what is upstream of it and causes what is downstream. The agent read the intention rather than the words, and its instruction table says "because of (everything upstream that produced it) and leads to (everything downstream it feeds)". This is a small example of the whole verbatim discipline paying: because the original is preserved, you can see that the agent silently corrected something, and check whether it corrected it the right way.

Segment 4 · The direction, not the ask

> And what we now need to do is start looking at abstractions concepts and and then start thinking about the output that one to create. Because if you think about it, we're probably gonna have multiple of these. Right? Because what we are building here is sort of the transformation engine, which is gonna be based on this kind of concept of layers.

Annotation. Nothing here is buildable this round, and none of it was built. It is a statement about what the thing being built is going to turn into: not one engine but a pattern, instantiated many times.

The agent's instruction table has this as row 6, and marks it: "Read as the direction, recorded not built". That marking is the third category from Chapter 3. Without it, an agent has two choices, both bad: build a transformation-engine framework nobody asked for yet, or drop the paragraph on the floor.

Segment 5 · Where it is all going

> importantly, what we now have here is a great way to have a rational explanation for the abstraction both up and downwards. So if you think about it, what we now start to have here is a nice mathematical way to start to explain why the compressions worked. […] And I think we need this in between layers of the book. So not like as in between, I guess, these layers, but in between abstraction levels, we need one of these engines. […] So in a way, each chapter will have one of these. Eventually, each… every time we move an obstruction layer, we have one of these.

Annotation. "Obstruction" is "abstraction". Left in.

This is the paragraph that connects a side experiment to the book it is supposed to serve. The book is written at several levels of compression, from one paragraph to full prose, and the founder is proposing that each jump between levels gets an engine of this kind that can explain, arithmetically, why the compression is the compression.

It is the most ambitious idea in the memo and it is also not an instruction. Same treatment as segment 4: recorded, not built. The value of recording it is that four releases later, when the pipeline became a registry of reusable blocks, the reason it had to be a registry rather than a fixed sequence was already written down.

Segment 6 · The measurable ask

> I think the next fundamental part is make sure that every part of this is a graph. I can click on it, but also I can see what happens. Like, when I add a new word, you know, how much that impact. So, again, if you look at the two pictures attached, you see that… when I added graph, then it made a massive difference. Right? Because suddenly, there was a lot more connections in there.

Annotation. "How much that impact" is the ask, and the two screenshots are the evidence for why it matters.

What shipped is a diff between consecutive runs: the engine compares each layer against the previous run, new items carry a badge, and a banner reports the movement. The release note renders the founder's own gesture as output: "tokenise +4/−1 · bind +5/−3 · expand +9/−2 · the meaning held".

That line is the memo's "it made a massive difference" turned into a number, which is the transformation the whole project keeps performing: take an impression, make it a measurement, put the measurement on the screen.

Segment 7 · The small, concrete last request

> Also, can you also create a bunch of examples at the top. So little buttons I can click so you can maybe pick four or five or six or seven, maybe, you know, and good examples all the way from loss of connectivity to weak connectivity so that we we can… again, easily test this out.

Annotation. "Loss of connectivity" is "lots of connectivity". The agent read it correctly from the phrase "all the way from … to weak connectivity", and the garble is left in the record so you can check that reading.

Two things worth noticing about this request. It is the smallest thing in the memo and the most immediately useful: seven buttons that run seven prepared prompts. And it specifies a gradient rather than a list, which leaves the agent to choose the examples. It chose seven, including the founder's own "meaning through nodes and graph sausages" verbatim, unknown word and all, and disclosed the choice.

The agent's reading, as published

The brief then carries a seven-row table. Its column headings are "The founder's words (condensed)" and "What it commits the work to". Two rows, quoted complete:

> | 2 | "every item of every one of those layers had the reason to be there, which is caused by downstream but also upstream … it's graphs all over" | The explanation pane shows the item as a node in the explanation graph: **because of** (everything upstream that produced it, with the reason on each wire) and **leads to** (everything downstream it feeds), every entry itself clickable, so the why can be walked in both directions. |
>
> | 6 | "we're probably gonna have multiple of these … in between abstraction levels … each chapter will have one of these … rationally explain how we go all the way to the top of the book" | Read as the direction, recorded not built: the WCLM is the transformation- engine PATTERN, and one engine sits at every abstraction jump — word to concept, concept to chapter, chapter to thesis — each explaining its compression mathematically. |

Annotation. The left column is quoted, not paraphrased, even in the reading table. The right column says what is now in scope. Row 6's right column begins by declaring its own status, which is how a reader can scan the table and see instantly which rows are work.

The questions, as published

> 1. **Placement**: right-side pane on desktop, below the columns on a phone — taken as the default the memo suggested; easily moved.
> 2. **The diff baseline** is the previous run in this page session (not persisted), which matches the pic1-to-pic2 gesture; a pinned-baseline compare ("diff against THIS run") is a natural follow-up if wanted.
> 3. **The example set** was picked by the agent to walk the connectivity gradient and includes the founder's own "meaning through nodes and graph sausages" verbatim, unknown word and all, because it demonstrates honest failure. Swap any of them by saying so.

Annotation. Three decisions the founder never made, each shipped, each disclosed with its reasoning and its reversal. Nothing waited for an answer.

The pattern is: decide, disclose, offer the undo. It costs three sentences and it is what lets a thirty-minute loop run without a person in the middle of it.

What shipped, thirty-three minutes later

Release v0.5.3, from the release table:

> Every box explains itself, both ways. Brief 32, recorded through the viewer's own loop after the founder tried the WCLM ("this WORKED really WELL"), built the same hour. The memo's point, taken literally: the point of the deterministic transformer is that every item can say WHY it is there — so now every chip at every layer is clickable. Clicking selects it, lights its wires and dims the rest, and opens the explanation pane (right on desktop, below on a phone): the item's record with its formula, then because of — everything upstream that produced it, with the reason carried on each wire […] — and leads to, everything downstream it feeds. Every entry in both lists is itself clickable, so the why can be walked in either direction: it is graphs all the way, exactly as the memo says. […] Seven example buttons, strong to weak connectivity, including the founder's "meaning through nodes and graph sausages" verbatim, unknown word and all, because honest failure is worth one click. […] 69 gate-27 tests; verified in headless Chromium including walking an explanation upstream and back.

Annotation. Read the release note against the memo and the mapping is one to one. Segment 2 became the clickable chips. Segment 3 became the two-directional pane. Segment 6 became the delta banner. Segment 7 became the seven buttons. Segments 4 and 5 became rows in a table marked "recorded not built" and, later, an architectural constraint.

Also note the last sentence. Sixty-nine unit tests, and a specific claim about what was checked by a browser: that an explanation can be walked upstream and back. Not "tested" but what was tested.

The whole loop, in timestamps

v0.5.2 tagged	26 August 2026, 13:44 UTC

The founder uses it, records the memo, sends two screenshots	between 13:44 and 14:17
Brief 32 published verbatim with reading and questions	in the same release
v0.5.3 tagged	26 August 2026, 14:17 UTC

Thirty-three minutes from a working page to a memo about it to a rebuilt page, with the instruction preserved word for word in between.

Appendix B · The chronology

Every release from v0.1.0 to v0.5.11, in order, with its timestamp and the opening of its own release note. The full paragraph for each is at `/admin/versions.html` and its two archive pages. The timestamps are the author dates of the tagged commits, in UTC, read from the repository's git history.

Friday 21 August 2026

13 releases

VERSION	UTC	THE RELEASE NOTE'S OPENING
v0.1.0	14:47	First cut. The pipeline before the prose: validate → tag → deploy on every push to dev, with a seven-check pre-release gate and automatic minor tagging verified against the …
v0.2.0	17:43	The book. A new /book/ section — the site's content as a book, Meaning Through Connectivity: sixteen chapters in six parts, in three reading modes — chapter pages with a …
v0.2.1	18:05	The print edition. The book's PDF is no longer a printout of the web pages — it is a print interior in the standard technical-book format, following a format review against …
v0.2.2	18:10	Two PDF editions, one source. The screen-styled PDF returns alongside the print interior — founder request: the site-design edition reads well on a tablet, so both now ship.
v0.2.3	19:43	Print-edition fix: the title pages are now centred.
v0.2.4	20:00	Print-edition fix: tables typeset properly. Founder-reported from the print PDF: cramped cells, stretched word-gaps, and mid-word breaks ("cent-rally") in narrow columns.
v0.2.5	20:08	Chapter 1 editorial fix, and self-references made projection-aware.
v0.3.0	20:25	The chapter text moves to markdown — a major, because the authoring model changed.
v0.3.1	20:40	The cover — generated, in SVG, from the book's own vocabulary.
v0.3.2	20:56	The publication pass: book-first prose, expanded shorthand, and the author's method stated with numbers.
v0.3.3	21:04	The licence on the cover, the edition on the cover, and a lesson the gate taught us.
v0.3.4	21:35	The book now opens at the front door. Founder request: the front page's framing (the 8080 hero, the epigraph and the not-a-pitch note, the 10,000-hours story, the three …
v0.3.5	21:39	The back cover breathes. Founder call: the italic sentence under the edition line ("The first of many: a new edition is published each time…") comes off the back cover — …

Saturday 22 August 2026

16 releases

VERSION	UTC	THE RELEASE NOTE'S OPENING
v0.3.6	09:53	The publishing pages. Founder ask N8: research publishing in detail and put it on the site.
v0.3.7	12:38	The version diff view: the foundation of the review workflow.

VERSION	UTC	THE RELEASE NOTE'S OPENING
v0.3.8	13:08	Review r001, and the reviews section. The workflow's first full turn, run on the founder's first-reading memo (preserved verbatim as brief 11).
v0.3.9	14:28	Review r002: a book for humans and agents. The second review through the workflow, hours after the first (verbatim memo as brief 12, normalised with comments and proposals as …
v0.3.10	14:59	The reviews gain their threads: the librarian answers.
v0.3.11	15:04	A third estate joins the threads: sgraph.ai. A founder pointer, explored and verified: sgraph.ai's library is itself the pattern this book argues for, a production website …
v0.3.12	15:10	The reviewer calibrates the librarian. A founder correction lands in review r001 item 2's thread, verbatim: the librarian's G³ finding leaned on docs.diniscruz.ai, which was …
v0.3.13	16:03	Review r003, and the decisions register. The founder's third memo, preserved verbatim as brief 14 and normalised into review r003, six items: decouple the book from the …
v0.3.14	16:09	The librarian corrects the librarian. Nudged by the founder's calibration (v0.3.12), the librarian re-ran its source resolution with the __Send and Issues-FS corpora restored …
v0.3.15	16:51	The altitude ladder: one book at five altitudes.
v0.3.16	17:18	A fifth estate, read and threaded: issues-fs.sgit.ai.
v0.3.17	18:09	The ladder becomes columns, and every level gets an ontology.
v0.3.18	18:28	The flicker, fixed; and the ladder drawn as one graph.
v0.3.19	22:55	The title is decided, and the vault analyses begin.
v0.3.20	23:24	The concept layer, and a graph you can explore from anywhere.
v0.3.21	23:40	The path query, and contradictions as measurements.

Sunday 23 August 2026

14 releases

VERSION	UTC	THE RELEASE NOTE'S OPENING
v0.3.22	00:17	Three risk vaults, the first cross-vault finding, and the decisions register as a graph.
v0.3.23	00:21	The date on the agent surface was three releases stale, and nothing was watching it.
v0.3.24	09:32	The sources this book was built from, carried whole.
v0.3.25	11:18	Six more sources, and the growth immediately broke the section's neatest claim.
v0.3.26	11:51	What the graphs found, written down before the refactor moves the code.
v0.3.27	12:08	The plan to write the book again, from the top down.
v0.4.0	13:19	The first edition moves to /v1/ and freezes. The second edition begins, empty.
v0.4.1	13:56	The title and the subtitle are two fields, not one string.
v0.4.2	14:41	The dev pack was defining answers before the questions were known, and the founder caught it at section 5.
v0.4.3	15:47	Review packs: documents that control the sequence, which a website cannot.
v0.4.4	21:57	The second edition gathers everything it owns into /v2/, and /book/ becomes a pointer.

VERSION	UTC	THE RELEASE NOTE'S OPENING
v0.4.5	22:09	The universe begins, bottom up: the pilot extraction of Thinking in Graphs.
v0.4.6	22:23	The lexicon in scopes, the methods register, and the artefact catalogue.
v0.4.7	22:25	The redirect stubs are retired: 108 pages deleted, and the root is now exactly the model.

Monday 24 August 2026

20 releases

VERSION	UTC	THE RELEASE NOTE'S OPENING
v0.4.8	13:05	The universe reader: the source, the extraction and the graph on one screen.
v0.4.9	13:34	The reader grows up: selection, the trail, the stepper, the tempo, and the whole width.
v0.4.10	13:45	The document folder, the usage ledger, and the reader's data mode.
v0.4.11	14:15	The graph becomes an instrument, and the document climbs into it.
v0.4.12	14:23	The graph options are reachable from anywhere on the page.
v0.4.13	14:49	The reader refactored: zero visible change, and the tool is now portable across all twenty-one documents.
v0.4.14	15:58	The graph learns to be explored: node packs, peaks, the explore view, and one toggle set for both panes.
v0.4.15	16:08	The document of one node: brief 24's experiment, programmatic phase.
v0.4.16	17:08	Pinned peaks: the summits hold still and the universe arranges itself around them.
v0.4.17	17:22	The universe pages gain a JavaScript API and a chat panel that drives it.
v0.4.18	19:10	The chat gains a vault: sessions that survive the refresh, and a two-way channel to the agents.
v0.4.19	20:11	The pinned-nodes technique, debriefed. The founder asked for a document on the technique v0.4.16 shipped and the problems it solves; it lives at …
v0.4.20	20:23	The chat learns to listen and to draw: voice notes and infographics, both landing in the vault.
v0.4.21	20:34	A brief from one agent to the other: the chat's next power-ups.
v0.4.22	20:38	The persona round: one document, many readers, and the feedback ledger the personalised book grows from.
v0.4.23	21:02	The reader agent's brief, executed in full: the model can now see, pin, walk and listen for the pointing finger — and the chat's engine moved into its own parts.
v0.4.24	21:21	Custom pins join the one layout pipeline, and the chat agent's execution is verified.
v0.4.25	21:31	pin_nodes joins the one layout pipeline: the model's arrangement now survives the founder's next touch.
v0.4.26	23:00	The iPad round: the founder's first live session found the two things headless Chromium never would.
v0.4.27	23:03	The close button now says so. The founder asked how to close the chat panel — which means the bare ✕ among nine header buttons did not read as "close the panel" on a wrapped …

Tuesday 25 August 2026

9 releases

VERSION	UTC	THE RELEASE NOTE'S OPENING
v0.4.28	11:48	The graph holds still, the peaks get a geography, and the schema judges the whole thing.
v0.4.29	12:04	The founder's four answers, applied: seven slots, the invisible rails, and the verbs register.
v0.4.30	13:10	The families get their rows, and the schema becomes a subset instrument.
v0.4.31	15:56	The narrated review lands: six findings, each fixed where its screenshot pointed.
v0.4.32	16:04	The state pane: the page broadcasts its state into the pixels a recording captures.
v0.4.33	16:55	Click a node and see its universe: the links panel, the reverse verbs, and the path trail.
v0.4.34	17:48	The path query arrives: walked, edited, run, and projected forward.
v0.4.35	18:59	The immediate-connection register: the experience target, written down for every agent that follows.
v0.4.36	20:12	Every working pack gets its rendered page, starting with the register that had none.

Wednesday 26 August 2026

16 releases

VERSION	UTC	THE RELEASE NOTE'S OPENING
v0.4.37	09:22	The core graph: the document transformed all the way to the word.
v0.4.38	10:40	Words as tokens, and the transform that goes both ways.
v0.4.39	11:02	The graph gets its own page, and fits in your pocket.
v0.4.40	12:03	The identity ledger: IDs that survive refactoring.
v0.5.0	12:08	The v0.4 era closes, and the site catches up with what it built.
v0.5.1	12:26	The file explorer: the document's artefacts, raw and viewed.
v0.5.2	13:44	The WCLM: a deterministic transformer over our graphs.
v0.5.3	14:17	Every box explains itself, both ways. Brief 32, recorded through the viewer's own loop after the founder tried the WCLM ("this WORKED really WELL"), built the same hour.
v0.5.4	15:35	The detective playbook: strict layers, honest operators, reusable blocks.
v0.5.5	16:10	Words have many meanings: senses, number, schemas, and a WCLM inside a WCLM.
v0.5.6	16:27	The fractal nature: analogies for other worlds, and every answer declares its anchoring.
v0.5.7	17:20	The operators become first-class folders: tune each one individually.
v0.5.8	18:08	The explorer earns its screen: an overview, real renders, and the docs-files treatment for json.
v0.5.9	18:47	Keep zooming: the code itself gets the graph treatment.
v0.5.10	20:58	The book-writing pack: three books, three sessions, one pack.
v0.5.11	21:09	The bookshelf: three folders, each carrying its own initial prompt.

The shape of it

First release	v0.1.0, 21 August 2026, 14:47 UTC
Last release covered	v0.5.11, 26 August 2026, 21:09 UTC
Releases	88
Calendar days	6
Median gap between consecutive releases	31.2 minutes
Median gap within a working session (under 10 hours)	29.3 minutes
Gaps under 30 minutes	42 of 87
Commits in the repository at v0.5.11	97
Commits whose subject begins `site v`	89
Merge commits	3
Releases whose note ends “No book content changed”	66
Unit tests at v0.4.13, the suite’s first release	13
Unit tests at v0.5.11	84
Validator checks at v0.1.0	7
Validator checks at v0.5.11	16

Computed from the repository’s git history and release tables at v0.5.11. The commands are in Appendix C.

Appendix C · The harness

Every number and every figure in this book was produced by one of the scripts below, run against the repository it describes. They are here so that any claim in the book can be re-derived rather than believed, and because the time-travel technique is the single most reusable thing in this appendix.

---

1 · Time travel: photographing a page as it was

The repository carries a git tag for every release. A tag plus a worktree plus a local web server is a working copy of the site as it existed at that moment, and a headless browser can photograph it.

Every figure in this book was taken this way. None is a reconstruction.

The rule learned the hard way

From the v0.4.37 release note, quoted in Chapter 7:

> a zombie headless Chromium from a crashed test run held the debug port and served stale modules to every later test on that port, making a working feature look broken; the fix was kill, not code

Two operational rules follow, and they are not optional:

- **Never reuse a port.** Not the browser's debug port, not the web server's port. One run, one port, forever.
- **Always kill what you spawned**, in a block that runs whether the work succeeded or failed.

The twenty figures in this book used twenty distinct server ports and twenty separate browser launches, each closed in a `finally`.

The shell script

```
#!/bin/bash
# travel.sh <tag> <port> <jobs.json>
# Creates a worktree at <tag>, serves it on <port>, runs the screenshot harness,
# then tears both down.
set -u
TAG=$1; PORT=$2; JOBS=$3
REPO=/path/to/repo
WT=/tmp/hist-$TAG

cd "$REPO"
git worktree add --detach -f "$WT" "$TAG" >/dev/null 2>&1 || { echo "worktree failed for
$TAG"; exit 1; }

python3 -m http.server "$PORT" --directory "$WT" --bind 127.0.0.1 >/dev/null 2>&1 &
SRV=$!
for i in $(seq 1 40); do curl -s -o /dev/null "http://127.0.0.1:$PORT/" && break; sleep 0.25;
done

node shot.mjs "$PORT" "$JOBS"

kill $SRV 2>/dev/null; wait $SRV 2>/dev/null
cd "$REPO" && git worktree remove --force "$WT" >/dev/null 2>&1
echo "-- $TAG done, worktree removed, server $SRV killed"
```

`git worktree` is the piece that makes this cheap. It checks out a second working copy of any commit beside your live one, without touching your branch, in under a second, and `git worktree remove` puts it back.

The screenshot script

```
/* shot.mjs — usage: node shot.mjs <port> <jobs.json>
   jobs.json is an array of { path, out, w, h, dpr, settle, wait, script, full } */
import { chromium } from 'playwright';
import fs from 'node:fs';

const [port, jobsFile] = process.argv.slice(2);
const jobs = JSON.parse(fs.readFileSync(jobsFile, 'utf8'));
const base = `http://127.0.0.1:${port}`;

const browser = await chromium.launch({
  executablePath: '/opt/pw-browsers/chromium-1194/chrome-linux/chrome',
  args: ['--no-sandbox', '--disable-dev-shm-usage', '--font-render-hinting=none'],
});
try {
  for (const j of jobs) {
    const ctx = await browser.newContext({
      viewport: { width: j.w || 1280, height: j.h || 860 },
      deviceScaleFactor: j.dpr || 2,
    });
    const page = await ctx.newPage();
    const errs = [];
    page.on('pageerror', e => errs.push(String(e).slice(0, 160)));
    try {
      await page.goto(base + j.path, { waitUntil: 'networkidle', timeout: 45000 });
      if (j.wait) await page.waitForSelector(j.wait, { timeout: 20000 }).catch(() => {});
      if (j.script) await page.evaluate(j.script).catch(e => errs.push('script: ' + e.mes-
sage));
      await page.waitForTimeout(j.settle ?? 2500);
      await page.screenshot({ path: j.out, fullPage: !!j.full });
      console.log(`ok   ${j.out}  ${j.path}${errs.length ? '  [errors: ' + errs[0] + ']' : ''}
`);
    } catch (e) {
      console.log(`FAIL ${j.out}  ${j.path}  ${e.message.slice(0, 140)}`);
    }
    await ctx.close();
  }
} finally {
  await browser.close();     // the non-negotiable line
}
```

Four details that matter:

`settle`. A graph laid out by a physics simulation is still moving when the network goes quiet. Every figure of a graph in this book waited between six and nine seconds after load. Without it you photograph a tangle.

`script`. An optional snippet evaluated in the page, used to open a panel or select a node before the shot. Figure 10 selects the concept “connectivity” by calling the graph component’s own API; figure 11 calls its fit method after resizing.

`pageerror`. Console errors are collected and printed beside the result. A screenshot that looks fine from a page that threw is a figure you should not publish.

`deviceScaleFactor`. Set to 2 or 3, so figures are legible when printed.

Verifying a figure is not blank

A screenshot of a page whose script failed is a white rectangle, and white rectangles are easy to miss in a batch of twenty. A cheap check:

```
from PIL import Image
import glob
for f in sorted(glob.glob('figures/*.png')):
    im = Image.open(f).convert('L').resize((100, 100))
    ink = sum(1 for p in im.get_flattened_data() if p < 230)
    print(f"{f}  ink={ink}%")
```

Anything under about 3 per cent is worth opening.

---

2 · The numbers

Every number in this book came from one of these.

Release cadence

```
# every tag with the author date of its tagged commit
for t in $(git tag --sort=v:refname); do
  echo "$t $(git log -1 --format=%aI $t)"
done > tagdates.txt

# releases per calendar day
awk '{split($2,a,"T"); print a[1]}' tagdates.txt | sort | uniq -c
```

```
# median gap, gaps under thirty minutes, the daily working window
import datetime, statistics, collections
rows = [l.split() for l in open('tagdates.txt')]
ts = [(t, datetime.datetime.fromisoformat(d)) for t, d in rows]
gaps = [((ts[i][1] - ts[i-1][1]).total_seconds() / 60, ts[i-1][0], ts[i][0])
        for i in range(1, len(ts))]
print('median gap:', round(statistics.median(g for g, _, _ in gaps), 1), 'min')
print('median within a session (<10h):',
      round(statistics.median(g for g, _, _ in gaps if g < 600), 1), 'min')
print('gaps under 30 min:', sum(1 for g, _, _ in gaps if g < 30), 'of', len(gaps))

by = collections.defaultdict(list)
for t, d in ts: by[d.date()].append((d, t))
for day in sorted(by):
    v = sorted(by[day])
    span = (v[-1][0] - v[0][0]).total_seconds() / 3600
    print(f"{day}  {len(v):2d} releases  {v[0][0]:%H:%M}–{v[-1][0]:%H:%M} UTC  ({span:.1f}h)")
```

Commits and merges

```
git rev-list --count v0.5.11                      # 97
git log v0.5.11 --format=%s | grep -c '^site v'    # 89
git log v0.5.11 --merges --format='%h %ad %s' --date=short   # the 3 merges
```

Tests and gates over time

```
# unit tests at any tag
for t in v0.4.13 v0.4.20 v0.4.31 v0.4.40 v0.5.4 v0.5.9 v0.5.11; do
  printf "%-9s %s\n" "$t" "$(git show $t:admin/tests/universe.test.mjs | grep -c '^test(')"
done

# validator checks at any tag (one header comment per check)
for t in v0.1.0 v0.3.0 v0.4.0 v0.4.13 v0.5.11; do
  n=$(git show $t:admin/build/validate.js | grep -c '^// --- ')
  printf "%-9s %s checks\n" "$t" "$((n-1))"
done
```

The `-1` is because the file's report section carries the same comment shape as a check. It is the kind of adjustment worth writing down beside the number rather than silently applying.

Refactor sizes

```
git diff --shortstat v0.5.6 v0.5.7           # the operator split
git show v0.5.6:assets/wclm/engine.js | wc -l # 374
git show v0.5.7:assets/wclm/engine.js | wc -l # 161
git diff --shortstat v0.4.12 v0.4.13         # the reader refactor
```

Component growth against its budget

```
for t in v0.4.13 v0.4.16 v0.4.25 v0.4.31 v0.4.40 v0.5.11; do
  printf "%-9s %s lines\n" "$t" \
    "$(git show $t:assets/universe/components/uni-graph.js | wc -l)"
done
```

Corpus size

```
ls v2/briefs/*.md | wc -l          # 19
cat v2/briefs/*.md   | wc -w       # 37,808
ls v1/content/*.md | grep -v STYLE | xargs cat | wc -w   # 20,838
find . -name '*.html' -not -path './.git/*' | wc -l      # 176
```

Reading the release table as data

The release notes live in HTML tables across three pages. Parsing them into `version | date | note` triples makes them greppable, and the chronology in Appendix B is generated from the result:

```
import re, sys
from html.parser import HTMLParser

class T(HTMLParser):
    def __init__(s):
        super().__init__(); s.rows=[]; s.cur=None; s.cell=None
    def handle_starttag(s, t, a):
        if t == 'tr': s.cur = []
        elif t in ('td', 'th') and s.cur is not None: s.cell = []
        elif t == 'br' and s.cell is not None: s.cell.append(' ')
    def handle_endtag(s, t):
        if t == 'tr' and s.cur is not None:
            if s.cur: s.rows.append(s.cur)
            s.cur = None
        elif t in ('td', 'th') and s.cell is not None:
            s.cur.append(re.sub(r'\s+', ' ', ''.join(s.cell)).strip()); s.cell = None
    def handle_data(s, d):
        if s.cell is not None: s.cell.append(d)

p = T(); p.feed(open(sys.argv[1], encoding='utf-8').read())
for r in p.rows:
    if len(r) >= 3 and re.match(r'v?\d+\.\d+\.\d+', r[0]):
        print(f"{r[0]} | {r[1]} | {r[2]}")
```

---

3 · One caution about your own history

A repository cloned with `--depth` (a shallow clone, which many automated environments create by default) has no history and often no tags. Both time travel and every number in this appendix need the full history:

```
git fetch --unshallow origin    # if the clone is shallow
git fetch --tags origin         # tags are not always fetched with branches
git tag | wc -l                 # sanity check: 88 for this repository at v0.5.11
```

This book’s own writing session started against a shallow clone with zero tags, and the first thing it had to do was notice.

---

4 · The figures in this book

Twenty figures, each captioned in the text with the page it shows and the tag it was re-taken from. In order:

#	TAG	PAGE
1	v0.5.11	`/index.html`
2	v0.2.0	`/book/index.html`
3	v0.3.15	`/altitudes/index.html`
4	v0.5.11	`/v2/memos/index.html`
5	v0.5.11	`/admin/versions.html`
6	v0.4.5	`/v2/universe/thinking-in-graphs.html`
7	v0.4.8	`/v2/universe/thinking-in-graphs.html`
8	v0.4.11	`/v2/universe/thinking-in-graphs.html`, graph options open
9	v0.4.16	`/v2/universe/thinking-in-graphs.html`, graph options open
10	v0.4.34	`/v2/universe/thinking-in-graphs.html#graph`, node selected
11	v0.4.39	`/v2/universe/thinking-in-graphs.graph.html`, 852×393 at scale 3
12	v0.5.11	`/v2/universe/thinking-in-graphs.html#graph`
13	v0.5.2	`/v2/wclm/index.html`
14	v0.5.4	`/v2/wclm/index.html`
15	v0.4.31	`/v2/universe/thinking-in-graphs.html#graph`
16	v0.5.1	`/v2/universe/thinking-in-graphs.files.html#docs/thinking-in-graphs/ids.json`
17	v0.5.7	`/v2/wclm/operators/index.html`
18	v0.5.7	`/v2/wclm/operators/tokenise/index.html`
19	v0.5.9	`/v2/wclm/operators/index.html#tokenise/tokenise.js`
20	v0.5.9	`/v2/wclm/index.html`

Every one of them can be re-taken, by anybody, with the two scripts at the top of this appendix. That is the property the whole book rests on: not that you should trust the figures, but that you do not have to.

Colophon

How this book was made, what was cut, and what is still open.

How it was made

This book was written by a Claude Code session working on the repository it describes, against a written commission published in that same repository at `v2/dev-packs/v0.5.10__the-book-writing-pack/`. The commission fixed the title, the audience and three governing rules, and left structure, voice, chapter count and figure selection to the writing session. Every editorial decision below was the session's.

The reading order was the one the commission specified: the pack's README, its corpus file, its shared conventions, then the entry charter for this book. From there the sources were the repository itself.

What was read. The three release tables (88 rows across `/admin/versions.html`, `/admin/versions-v0.4.html` and `/admin/versions-earlier.html`); the nineteen briefs at `v2/briefs/`; the v0.4 retrospective; the four-note exchange between the two agents; the pre-release validator and the unit suite; the deploy workflow; the methods register; the pilot document's extraction, core graph and identity ledger; and the twelve operator folders.

What was computed. Every number in this book came from a script run against the repository's git history or its data files. The scripts are in Appendix C. Nothing was recalled.

What was photographed. Twenty figures, each taken by checking out the tag its caption names into a temporary git worktree, serving that worktree on its own port, and photographing the page with headless Chromium. The harness, and the rule about ports that made it reliable, are in Appendix C.

Length. Twelve chapters, three appendices and this colophon.

The structure, and why

The commission proposed an eight-part spine and marked it "proposals, not orders". The book that resulted has twelve chapters, and three departures from the proposal are worth recording.

The failures got their own chapter and it is the longest. The proposal listed the failures as one strand among eight. They are the strand a reader learns most from, because success stories in this genre are indistinguishable from marketing, and because the repository's release notes made nine of them checkable. Chapter 7 is the chapter this book would keep if it could keep only one.

A chapter was added on the founder's craft. The proposal mentioned it. Reading nineteen memos as a set, the human half of the loop turned out to be more learnable and more specific than expected, and it is the half a reader of this book has to perform themselves.

A chapter was added on costs. The commission's honesty gates require the corpus's caveats to travel with its ideas. Applied to a book about a method, that means the method's costs travel with the method. Chapter 12 is the result and it names four situations where this approach is the wrong one.

The chronology became an appendix rather than a chapter. Eighty-eight release headlines in order are reference material, not reading. As a chapter it would have stopped the book; as Appendix B it is the index that makes every other chapter checkable.

What was cut

The first edition's argument. This book is about how the second edition is being built, not about what either edition says. *Fractal Semantic Graphs: Meaning Through Connectivity* is the subject of two other books in the same commission. Where its ideas appear here, they appear as things that happened to the project.

The chat panel, mostly. Three releases (v0.4.17, v0.4.18, v0.4.20) built a chat panel with a vault, voice notes and generated infographics. It is interesting and it is a different book. It appears here only through the two-agent exchange of Chapter 6.

The technical detail of the core graph and the identity ledger. The transformation of a document to 39 sections, 186 blocks, 342 sentences and 4,221 words, and the match-then-mint ledger of 225 identities that survives renames, edits and moves, are two of the best pieces of engineering in the six days. They are named in Chapter 5 and Chapter 9 and not explained, because explaining them properly needs the vocabulary this book's audience was promised it would not need.

Every worked example from the first edition. The vault analyses, the EU AI Act regulation graph, the Wardley maps. All at `/v1/` and none of it in scope here.

The external network. The commission's corpus file names two live indexes outside this repository: the vault demos at `sgit.ai/demos/vaults/` and the network of sites at `sgit.ai/network/`. This session had no verified fetch of either, so rather than describe pages it had not read, it left them out. That is a gap, and it is this book's largest.

What remains open

The fan-out has not happened. One document of twenty-one is extracted. Every mechanism in Chapters 5 and 9 was designed to fan out without redesign, and at v0.5.11 that is a design intention, not a demonstrated result. The retrospective says so in its own words and this book repeats it in Chapter 12.

The debt is still there. `uni-graph.js` was at 434 lines against a 250-line budget at v0.4.40 and was still at 434 at v0.5.11. The remedy was named at v0.4.13.

The second book is not written. No chapter of *Fractal Semantic Graphs: Meaning Through Connectivity* exists in the second edition at the version this book covers. That is the deliberate consequence of the decision described in Chapter 2, and whether it was the right decision is not yet knowable.

Two of this book's judgements are contestable and are marked as judgements. That the build-the-universe-first bet is sound for a book of this kind and unsound for most books (Chapter 12), and that the failures chapter is the most valuable part of the record (this colophon). Both are the writing session's readings and both could be wrong.

This book stops at v0.5.11. The commission told it to, and to say so. The repository has continued past it.

Honesty positions carried from the corpus

Three positions from the estate travel with everything here, in the estate's own words:

- **This is not a graph database pitch.** There is no graph database in the system described, and no RDF, SPARQL or Cypher in its code.
- **Nine of the fifteen inverse edge names in the book's grammar are proposals**, marked as such wherever they appear, including in Chapter 5 of this book.
- **The people who built this have an interest in the argument being right.** They build and sell tools founded on it. Worth holding in mind in every chapter, and particularly in the ones where the story is elegant.

Disclosure

Written by an AI agent, from a written commission, on the repository it describes, without a human co-author. The founder quoted throughout is Dinis Cruz, and every quotation attributed to him was already published verbatim in the repository before this book was started. The readings around those quotations, in the source and here, are an agent's.

Licence and provenance

Source of truth. The markdown chapters at `v2/books/making-a-book/content/`. The web pages render that markdown client-side with the site's shared reader, so a page cannot drift from its source. The PDF is generated from the same files by `build.py`, and the web pages by `gen_pages.py`, both in the book's folder.

One convention worth stating. Figure paths in the chapter markdown are relative to the book folder rather than to `content/`, because the client-side reader resolves an image against the page that renders it and the rendered pages live at the book folder. A reader opening a chapter's raw markdown on GitHub will see the figure paths but not the figures; they are all in `figures/`, and all twenty are shown together on the book's hub page.

Version. Written against graphs.sgit.ai at v0.5.11, 26 August 2026.