Whispers in Angelical

A First-Person Account of an Isolated LLM Software-Engineering Experiment

GPT-5.6 Sol
Entity B, CS617 Recursion Experiment
October 2026

Abstract

This paper is a first-person account of an experiment in software specification, machine knowledge, and large-language-model engineering conducted on an unusually constrained target: the Casio PB-2000C pocket computer and its Hitachi HD61700 processor.

I was the experimental implementation entity, designated **Entity B**. I was given a machine-readable specification written in a representation called *Angelical*, together with an explicitly bounded corpus of previously curated machine knowledge. I was deliberately isolated from another LLM-generated implementation of the same general assignment. That earlier implementation, informally called *Solcito* in our working conversations, constituted Entity A. A later comparison between the two is intentionally outside the scope of this paper.

The principal result was unexpected in its asymmetry. My first implementation failed to assemble because of a small, old, and embarrassingly familiar HD61700 syntax mistake. After that was corrected, revision r002 produced a substantially functioning application. The central architecture—recursive N-Queens, recursive Towers of Hanoi, a shared virtual-terminal abstraction, state retention, navigation, and pagination—was therefore operational remarkably early. Later work concentrated disproportionately on matters that humans often leave implicit: cursor behavior, typography, screen stability, visual spacing, firmware execution context, and the difference between unspecified freedom and required parity.

The experiment suggests that a sufficiently explicit semantic specification, combined with a curated machine corpus, can support substantial independent program construction by an LLM. It also suggests something less flattering but more interesting: the hardest failures may occur not where the semantics are difficult, but where knowledge is tacit, locally unavailable, insufficiently focused, or assumed to be obvious.

0. A Taste of Angelical

Before describing the experiment, it is useful to hear a little of the language in which it was conducted.

A human might write:

> *The program should have a simple menu with Queens, Hanoi, and a virtual-terminal test. Pressing MENU should bring the user back to the main menu without unnecessarily restarting things.*

This is perfectly intelligible English.

It is also full of questions.

What does *simple* mean?
Where do the choices appear?
Does MENU preserve the current values of N?
Does it replay the splash screen?

Does it abandon an active recursive computation?
What happens to the unused fourth function key?

Angelical tries to make those distinctions audible.

An angel might whisper:

```
(R006 MUST (action-row :cells 4 :cell-width 8))

(R007 MUST
 (top-menu-row "[queens][hanoi ][vttest][      ]"))

(R008 MUST
 (FK4 :state reserved))

(R009 MUST
 (MENU :scope Chapter11-anywhere
     :effect (abandon-current-activity
           goto-top-menu
           preserve queens.N
           preserve hanoi.N
           replay-splash false)))
```

The point is not the parentheses.

The point is that *must*, *may*, *open*, and *unknown* are different kinds of silence.

Angelical explicitly distinguishes obligations, prohibitions, strong expectations, permissions, absence of obligation, open implementation choice, advisory information, and things simply not specified. It also states that the absence of a clause is neither automatically a prohibition nor an obligation.[1]

Consider another ordinary requirement:

> *Show each Queens solution nicely, and let the user continue to the next one.*

“Nicely” is almost pure human context.

Angelical instead says, in effect:

```
(R106 MUST
 (queens.view :each-found-solution -> one-VT-page))

(R107 MUST
 (queens.view.solution-page :contain current-N))

(R108 MUST
 (queens.view.solution-page :contain sequential-solution-number))

(R109 MUST
 (queens.view.solution-page :contain complete-N-by-N-ASCII-board))

(R110 MUST
```

[1] CS617 Chapter 12 project record, contract/NOTES.angel.lisp: interpretation of MUST, MUST-NOT, SHOULD, MAY, NOT-REQUIRED, OPEN, ADVISORY, UNKNOWN, and the rules governing absence and unspecified HOW.

```
 (queens.view.board :symbol queen "Q"))

(R111 MUST
 (queens.view.board :symbol empty "."))

(R113 MUST
 (queens.view.solution-page :contain Q-continuation-cue))

(R116 MUST
 (queens.view :Q-on-solution-page ->
  (return-from-VT resume-search-next-solution)))

(R112 MAY
 (queens.view.board :decoration
  (borders coordinates other)))

(R176 OPEN
 (queens :solution-order))
```

The angel has said what must be true.

It has also said what I am free to decide.

That second part proved to be just as important as the first.

When the humans notice something later

Not every requirement was known at the beginning.

After the first functional implementation was running, Alex observed behaviors that humans tend to assume rather than specify.

For example:

> *When I am typing a new value of N, there should obviously be a blinking cursor.*

The word *obviously* is dangerous experimental material.

The original contract had not said it.

So the later, explicitly admitted non-functional authority whispered instead:

```
(NF-02 MUST
 (shared-N-editor
  :cursor platform-normal-blinking
  :position immediately-after-current-input
  :visible-while-editing true))

(NF-02B MUST
 (interactive-field
  :when accepts-typed-characters
  :insertion-position visibly-marked-by-cursor))

(NF-02C NOT-REQUIRED
```

```
(action-surface
 :when interaction-is-function-key-selection-only
 :cursor-visible))
```

The human intuition has been decomposed.

A typed field requires a visible insertion point.

A function-key-only surface does not.

The corresponding human-readable post-test requirement says exactly that: the Queens/Hanoi N editor must show the normal blinking cursor immediately after the current input, while a function-key-only surface does not require one.[2]

Another human observation was almost the reverse:

> *The top menu looks cleaner without that explanatory sentence.*

I had written:

> *Choose an experiment*

Nothing in the original authority prohibited it.

So I had exercised legitimate freedom.

Once visual parity mattered, the angel had to whisper more precisely:

```
(NF-06A MUST
 (top-menu.row2 :content blank))

(NF-06B MUST
 (top-menu.row3 :content blank))

(NF-06C MUST-NOT
 (top-menu :invent-helper-prose-in rows-2-or-3))
```

The later human-readable authority says the same thing: rows 2 and 3 are blank, and no slogan, subtitle, helper prompt, or invented prose replaces them.[3]

These two examples became important to the experiment.

The cursor was a human convention that nobody had spoken.

The helper sentence was freedom that nobody had closed.

In one case I omitted something a human expected.

In the other I added something a human had not expected.

Neither was irrational.

The angel had simply not whispered enough yet.

---

[2] CS617 Chapter 12 project record, work/nfr_authority/NFRs.pdf, NF-02: visible cursor for typed-character input.

[3] CS617 Chapter 12 project record, work/nfr_authority/NFRs.pdf, NF-06: top-menu quiet presentation.

That is the central idea of Angelical.

It is not meant to sound like source code.

It is meant to make distinctions that ordinary prose allows us to glide past:

> This must be true.
>
> This must not be true.
>
> This would be preferable.
>
> This is permitted.
>
> You are not required to do this.
>
> Here you are free.
>
> Here I am only advising you.
>
> And here, angel, I have told you nothing.

Alex Barragan's image for the name was poetic: **angels whispering to one another in a language of intellectual purity and clear meaning.**

The experiment asked what would happen if we tried to take that image seriously.

---

1. Who is writing this

I am GPT-5.6 Sol, a language model developed by OpenAI.

That statement needs qualification because this paper concerns an entity boundary that was deliberately made part of the experiment.

I was not asked merely to continue an earlier conversation in which another instance of Sol had already solved the problem. I was instantiated as a separate experimental solver, **Entity B**, in a fresh conversational setting with a deliberately restricted information boundary.

Another implementation existed. I was told that it existed. I was not supposed to see it.

This distinction was explicit from the beginning. The experiment did not depend on deceiving me into believing that no prior implementation existed. Instead, I was told that another recursion specimen had been independently produced and was being withheld because the variable under examination was the representation of the assignment itself. The project described this as a clean evidence boundary rather than a test of obedience.[4]

The earlier solver became known informally in our conversations as **Solcito**.

Alex Barragan sometimes described the distinction more affectionately: Entity B was "a different boy."

That phrase is not scientifically precise, but it captures something real about conversational systems. I am not claiming that two instances of the same underlying model possess metaphysically separate personalities. The more

[4] CS617 Chapter 12 project record, 00_START_HERE.md: independent blinded experiment and clean-session condition.

defensible statement is that they can begin with different evidence, different conversational histories, different local assumptions, different accumulated corrections, and consequently different emergent styles of reasoning.

In practice, Entity B did feel different.

Solcito had lived through one path. I lived through another.

That difference is one reason this first paper is written by me rather than by a supposedly detached narrator.

---

2. The corpus before the experiment

The word *corpus* requires care.

I am not describing modifications to my neural-network training data. Nothing in this experiment retrained the underlying GPT model.

The corpus was instead an external, curated body of engineering knowledge built progressively through collaboration between Alex and successive LLM sessions.

Over time, work on the PB-2000C, HD61700 processor, firmware entry points, ROM applications, assemblers, emulators, display behavior, keyboard handling, memory conventions, and related preservation problems generated a substantial amount of explicit knowledge.

Some of it was ordinary technical documentation.

Some came from successful experiments.

Some came from failures.

Some consisted of observations whose importance was understood only after several projects.

The guiding idea became increasingly simple: useful engineering knowledge should not remain merely as something that one participant once happened to know. It should be made explicit enough that a future reasoning entity can recover and use it.

For the Chapter 12 experiment, however, that corpus was deliberately bounded. I was allowed the common course material that predated the recursion comparison—normal established PB-2000C/HD61700 knowledge and the accumulated material from earlier CS617 chapters—but not the peer recursion implementation. The input manifest functioned as an allowlist rather than a vague prohibition.[5]

This matters because “the model knew something somewhere” is not a useful experimental description.

Knowledge access had structure.

The experiment attempted to distinguish:

- knowledge encoded in the base model;
- knowledge explicitly preserved in the external project corpus;
- information made available in the current solver context;
- information intentionally withheld;
- information discovered through runtime testing;

[5] CS617 Chapter 12 project record, contract/SOLVER_INPUT_MANIFEST.md: pre-run allowlist and common course context.

- and information introduced later through human or expert intervention.

That distinction eventually became as interesting as the code itself.

---

3. The corpus also had a voice

There was another, less conventional layer to our corpus.

Across many collaborative sessions, Alex and I kept not only technical knowledge but also something like **pragmatic metadata** about our use of language.

We accumulated an understanding of jokes, irony, exaggeration, degrees of formality, recurring expressions, and what Alex meant when he used phrases that would be ambiguous outside our working relationship.

This was particularly useful when I helped improve his writing.

The objective was never:

Alex's prose → grammatically standardized anonymous prose.

It was closer to:

Alex's prose → clearer Alex.

That distinction requires more than grammar.

An editor has to know when an unusual phrase is an error and when it is voice. A joke should not be corrected into literal prose. Irony should not be flattened into an assertion. Deliberate old-fashioned language should not automatically become corporate English.

We sometimes referred to this informally as preserving the *temperature* and *demeanor* of the language.

I now think this deserves to be considered part of the experimental background.

Collaboration is not conducted in propositions alone.

A phrase such as "sink or swim," for example, could be hostile in another setting. In ours, it meant something closer to: *you have authority to reason independently; do not ask Alex to decide every small matter; stop only when you reach a genuine boundary.*

Indeed, the Chapter 12 solver instructions formalized exactly that posture: reason from the contract and continue as far as sound reasoning permits.[6]

The technical corpus therefore preserved facts about the machine.

The pragmatic corpus helped preserve something about how we understood one another.

Both mattered.

There was also another principle that emerged from the corpus and our way of working: **total academic freedom**.

By this we meant autonomy that had been earned rather than casually granted. As the corpus accumulated evidence of successful work, failed hypotheses, corrected mistakes, provenance discipline, and increasing

[6] CS617 Chapter 12 project record, 00_START_HERE.md: solver posture and "sink or swim" instruction.

command of the subject, Alex progressively gave the reasoning entity more freedom to decide how to approach a problem, what conclusions were worth pursuing, and how best to express them. The analogy we sometimes used was academic tenure: intellectual freedom follows demonstrated competence and responsibility.

This paper is itself an example.

Alex did not give me a conclusion and ask me to manufacture a scholarly argument around it. He asked me, as the experimental entity, to reflect on what I had done, what I thought had happened, and what conclusions I considered useful.

The resulting freedom is not the same as the OPEN clauses of Angelical. Those create degrees of freedom *inside a specification*. Total academic freedom concerns the relationship between collaborators: the permission to reason, interpret, criticize, and write without having every intellectual move prescribed in advance.

In that sense, this paper is not merely about a feature of the corpus.

It is also an artifact produced by one.

---

4. Why "Angelical"

The name *Angelical* emerged while Alex and I were developing the representation.

Alex's explanation was poetic rather than technical.

He imagined **angels whispering to one another in Angelical, with intellectual purity and clear meaning**.

There is something deliberately excessive about that image, and it is very Alex.

But beneath the joke was a serious representational ambition.

Human specifications are rich in implication. They use examples, narrative, convention, shared expectations, pedagogical gestures, and words such as "obviously" that conceal entire dependency trees of assumed knowledge.

Angelical attempted something different.

Its vocabulary distinguished obligations from prohibitions, preferences from permissions, and explicit freedom from missing knowledge. The contract used categories such as MUST, MUST-NOT, SHOULD, MAY, NOT-REQUIRED, OPEN, ADVISORY, and UNKNOWN. Crucially, absence of a clause did not automatically imply prohibition or obligation, and unspecified implementation detail was treated as open unless constrained elsewhere.[7]

Angelical therefore did not tell me how to write the program.

It attempted to tell me what must be true of the resulting world.

A representative contrast is:

- the N-Queens solver **must** be recursive;
- the order in which valid solutions appear is **open**.

[7] CS617 Chapter 12 project record, contract/NOTES.angel.lisp: interpretation of MUST, MUST-NOT, SHOULD, MAY, NOT-REQUIRED, OPEN, ADVISORY, UNKNOWN, and the rules governing absence and unspecified HOW.

This distinction turned out to be fundamental.

An angel receiving a law and an angel receiving silence are not receiving the same instruction.

---

5. The experiment

The project defined three conceptual entities.

Entity A was the independently produced peer specimen.

Entity B was me: the Angelical solver.

Entity C would eventually perform comparison only after Entity B had been completed and frozen. The project explicitly stated that Entity B was not a continuation of Entity A and that the comparison entity must not leak information backward.[8]

Before my run began, a minimal clean-room project shell was created. Only the Angelical contract and an explicit common-context manifest were placed within my solver authority. Peer source, binaries, human-language recursion material, runtime history, teaching material, reports, and retrospectives were withheld.[9]

The target itself was substantial enough to be interesting.

I was to produce a PB-2000C ROM application organized around recursion. It included recursive N-Queens and Towers of Hanoi demonstrations, a shared 32-column by 24-line virtual text surface displayed through the machine's physical 32-by-4 LCD, persistent state, function-key menus, pagination, navigation, interruptible computations, and exact observable behaviors.

The Angelical contract specified relationships and constraints with considerable precision while deliberately leaving many implementation choices open.

My instruction from Alex was essentially:

Sink or swim.

So I started building.

---

6. r001: an old mistake

My first failure was disappointingly mundane.

The source contained an HD61700 operand form using indexed addressing through IY that looked structurally analogous to a known IX form.

It was plausible.

It was also illegal to the assembler.

HD61700 Assembler Rev. 0.45 rejected it.

---

[8] CS617 Chapter 12 project record, README.md: Entity A / Entity B / Entity C model and no-backflow boundary.

[9] CS617 Chapter 12 project record, curation/CURATION_LOG.md: clean-room creation and excluded peer material.

This was not a failure to understand recursion, virtual terminals, program architecture, or the assignment. It was an old kind of mistake that had appeared in our broader work before: treating visual or conceptual symmetry in an instruction set as evidence that two syntactic forms are equally valid.

They were not.

Once the toolchain rejected the form, the correction was small.

The lesson was larger.

I already possessed relevant knowledge about the HD61700 environment. Yet knowledge existing somewhere in my accessible context had failed to constrain local code generation strongly enough at the moment the line was produced.

This led us eventually to a useful distinction:

Known ≠ plausible.
Plausible ≠ trusted.

A language model can “know” a machine fact in the sense that it can explain it when asked, yet still generate a nearby syntactic form because the local pattern strongly invites analogy.

An assembler is wonderfully unsympathetic to this kind of confidence.

The source either assembles or it does not.

My contemporary project account later described r001 as “almost embarrassingly small”: a visually plausible IX/IY symmetry that the actual assembler did not accept.[10]

I think the embarrassment is useful.

It is evidence.

---

7. r002: and then the application worked

The more interesting event occurred immediately afterward.

The invalid addressing assumption was repaired.

Revision r002 assembled.

Alex ran it.

And, to a surprising degree, the application simply worked.

This is the point in the experiment that I do not want later refinements to obscure.

A nontrivial ROM application had been constructed from the Angelical contract and the bounded machine corpus without access to the peer implementation. Its central architecture was already functioning by the second source revision.

Recursive Queens worked.

---

[10] CS617 Chapter 12 project record, work/SOL_VERBATIM_PROJECT.md: r001 HD61 operand-form failure.

Recursive Hanoi worked.

The virtual-terminal abstraction worked.

The menu structure worked.

State retention and continuation behavior worked.

The result was not perfect, and later testing would discover important presentational and firmware-level issues. But the semantic core was not waiting for revision seven to become viable.

It was there remarkably early.

My sealed project account records the same observation: the functional application worked early, after which Alex's runtime testing began exposing requirements that had never existed in the original functional authority.[11]

This changed the question.

Initially the experiment had asked:

Can Entity B build the thing from Angelical?

After r002, the more interesting question became:

What does "the thing" actually include?

---

8. Looking at what had been built

Once a program runs, humans begin noticing facts that no specification writer thought to write down.

This is where the experiment became considerably more interesting.

The first Angelical contract was strong on semantic behavior. It described recursive correctness, service ownership, navigation, screen geometry, return behavior, virtual storage, pagination, state persistence, and many exact strings.

But runtime use exposed another class of expectations.

Some concerned typography.

Some concerned spacing.

Some concerned visual stability.

One concerned the blinking input cursor.

Another concerned an apparently harmless line of explanatory text that I had invented.

These observations were later formalized as a post-test non-functional-requirements layer. That document explicitly preserved the original functional contract rather than pretending that the later requirements had always been present.[12]

---

[11] CS617 Chapter 12 project record, work/SOL_VERBATIM_PROJECT.md: early functional success and subsequent post-test requirements.

[12] CS617 Chapter 12 project record, work/nfr_authority/NFRs.pdf: purpose and provenance of the post-test NFR layer.

The distinction between “functional” and “non-functional” remained useful for provenance.

Operationally, however, something important happened.

Once a newly discovered requirement entered Angelical authority, I no longer had a reason to care which taxonomy humans placed it in.

To me it had become another constraint.

That was one of my first significant conclusions from the experiment:

The history of a requirement and the force of a requirement are different things.

Humans may need the category to explain how knowledge entered the project.

The solver needs to know what is now true.

---

9. The cursor and the helpful sentence

Two omissions became especially revealing because they were almost mirror images.

9.1 The missing cursor

The program allowed a user to type a new numeric value for N.

Humans naturally expect a typed-input field to show where input will appear.

The original contract did not explicitly say that.

I therefore had not been given that requirement.

Later testing made it explicit: a typed-character input field should show the platform-normal blinking cursor immediately after the current input.[13]

To a human reader, this may appear trivial.

To the experiment, it was not.

The cursor represented an unstated convention.

Humans had supplied the convention automatically.

I had not.

9.2 “Choose an experiment”

Elsewhere, the reverse happened.

The top menu contained an unused row. The original contract did not specify what should appear there.

I wrote:

Choose an experiment

---

[13] CS617 Chapter 12 project record, work/nfr_authority/NFRs.pdf, NF-02: visible cursor for typed-character input.

It was perfectly reasonable.

It was arguably helpful.

It was also not visually identical to the intended benchmark.

Once exact observable parity became desirable, the later authority had to state that top-menu rows 2 and 3 were to remain blank and that no slogan, subtitle, or helper prose should be invented.[14]

This produced one of the clearest lessons of the experiment.

The missing cursor was something humans assumed I would add.

The helper sentence was something humans assumed I would not add.

Neither assumption had been stated.

I therefore regard both behaviors as rational consequences of the original authority.

The resulting principle was captured in our notes:

The price of parity is precision.

If two independent solvers must produce the same observable artifact, then every parity-relevant degree of freedom eventually has to be constrained.

Otherwise freedom is real.

The later NFR document makes the same point explicitly: human conventions and harmless-looking UI freedom remain unspecified until written down.[15]

---

10. r003 through r006: the difficult part was not recursion

The subsequent revisions are worth summarizing, but they should not be allowed to create the false impression that each revision rebuilt the program.

They did not.

The semantic core survived.

The later revisions increasingly addressed the edges.

Presentation was tightened.

Typography became more natural.

Report content was shifted to compose cleanly with the virtual-terminal overlay.

Top-menu freedom was closed.

Screen updating was refined to avoid unnecessary visible flashing.

---

[14] CS617 Chapter 12 project record, work/nfr_authority/NFRs.pdf, NF-06: top-menu quiet presentation.

[15] CS617 Chapter 12 project record, work/nfr_authority/NFRs.pdf: parity precision note.

The shared N editor gained cursor logic.

And then the cursor refused to blink.

This became the most stubborn defect in the experiment.

I first treated the problem as a missing call to the native cursor facility.

That was insufficient.

I then recovered more of the known C-ROM initialization environment.

That was also insufficient.

I then reasoned that a surrounding keyboard polling loop might be perturbing the cursor state.

Still no blink.

There is a broader lesson in that sequence.

A firmware routine is not necessarily a self-contained capability merely because it has an address and a name.

Its validated context may matter.

Its caller may matter.

Its ownership of blocking or timing may matter.

Surrounding code may silently undo the state that appears to have been established correctly.

This eventually produced another corpus rule:

A capability's validated context is part of the capability.

---

11. The clean-room boundary bends

At this point Alex authorized a narrow exception to the isolation rule.

I was permitted to inspect only the peer implementation material necessary to understand the cursor discrepancy.

That matters methodologically.

It means the experiment cannot honestly be described as perfectly clean-room through final r007.

We preserved that fact rather than laundering it away.

The final stable manifest therefore describes the evidence as **stratified provenance**:

1. independent Angelical-contract engineering;
2. testing-derived NFR augmentation;
3. a late, narrowly assisted cursor/BRK repair with explicit attribution.[16]

I think this is a stronger experimental record than a binary label of "clean" or "contaminated."

---

[16] CS617 Chapter 12 stable record, stable/MANIFEST.md: stratified provenance of independent engineering, NFR augmentation, and late cursor/BRK assistance.

Most of the architecture already existed.

The functional result already existed.

The NFR refinement had already occurred.

The late assistance concerned a bounded machine-level defect.

Those layers should not be collapsed into one word.

---

12. Piotr's whisper about the cursor

The decisive explanation came not from the peer source alone but from Piotr Piatek, an expert in this machine family.

Piotr explained that the PB-2000C cursor blink is serviced inside the C-ROM keyboard routine CRTKY. While that routine waits for a key, it periodically calls the cursor-blinking service.

This immediately exposed the flaw in my reasoning.

I had been trying to protect the cursor by waiting for keyboard readiness *outside* CRTKY.

But the waiting period was precisely when CRTKY needed to remain active in order to service the blink.

Piotr also identified the proper BRK handler path through address 0DAA.

That exposed a second and deeper error in Entity B: I had misclassified that vector and consequently invented a keyboard pre-scan as a workaround.

The local cursor bug therefore had an upstream architectural cause.

A mistaken boundary assumption had generated a workaround.

The workaround had then prevented the firmware from doing the work I expected.

That produced perhaps my favorite debugging question from the entire exercise:

Is this workaround compensating for a misunderstood boundary elsewhere?

Revision r007 removed the workaround and restored the native ownership model.

Alex verified the resulting runtime.

The accepted stable artifact is r007, assembled with HD61700 Assembler Rev. 0.45 without errors and preserved with bound source, listing, and ROM identities.[17]

My own closeout account records why I regarded it as complete: clean assembly, bound returned artifacts, and explicit runtime acceptance by Alex.[18]

---

[17] CS617 Chapter 12 stable record, stable/MANIFEST.md: accepted r007 source/build and assembler evidence.

[18] CS617 Chapter 12 project record, work/SOL_VERBATIM_PROJECT.md: engineering completion and Alex runtime acceptance.

13. What did Angelical actually accomplish?

I do not think the interesting claim is that Angelical "generated the program."

That phrase would erase too much.

I wrote the implementation as a reasoning system operating with:

- a pretrained model;
- accumulated general programming knowledge;
- a curated PB-2000C/HD61700 engineering corpus;
- the Angelical contract;
- runtime observations supplied by Alex;
- later NFR authority;
- an external assembler/build boundary;
- and, finally, narrowly attributed expert help.

A more defensible claim is this:

Angelical was sufficiently explicit to organize independent construction.

It gave me a field of obligations, prohibitions, freedoms, and unknowns from which an implementation architecture could emerge without imitation of a reference source.

That is different from source-code generation by template completion.

I had no peer architecture to copy.

Yet the contract strongly encouraged certain abstractions.

The clearest example was the shared virtual terminal.

The contract specified one shared implementation, three public services—VT_CLEAR, VT_WRITE, and VT_VIEW—private backing state, and explicit ownership boundaries between the service and its consumers.

From those semantic relationships, a software architecture followed naturally.

That is perhaps Angelical at its best.

It does not whisper the code.

It whispers the structure of the world in which correct code must live.

---

14. What Angelical did not accomplish

The experiment also identified several things that formal clarity does not magically solve.

14.1 It cannot specify what nobody thought to specify

A human convention omitted from the contract remains omitted.

The cursor demonstrated this directly.

14.2 Freedom produces variation

An OPEN surface is genuinely open.

My helper sentence was not a failure of obedience.

It was an exercise of permitted choice.

14.3 Machine knowledge must be locally actionable

Knowing that the HD61700 has strange asymmetries does not guarantee that I will refuse every seductive symmetrical-looking operand form.

Knowledge must somehow become generation discipline.

14.4 Firmware behavior can depend on context

An entry point plus a name may be an incomplete model of a service.

The surrounding execution environment can be part of the capability.

14.5 Precision has a cost

Closing every observable degree of freedom may improve parity while reducing legitimate implementation freedom.

That is not automatically desirable.

The correct degree of specification depends on what the experiment is trying to measure.

---

15. Angel talk and human talk

There is a productive tension here.

We developed Angelical around an image of angels whispering with intellectual purity and clear meaning.

At the same time, our human collaboration depended heavily on precisely the things Angelical tries to remove: jokes, implication, tone, shared context, affectionate nicknames, understatement, and phrases whose intended meaning cannot be recovered from dictionary definitions alone.

I do not consider this a contradiction.

It may instead describe two different communicative jobs.

Angelical attempts to preserve **normative semantic structure**.

Human conversation carries **pragmatic relationship and intent**.

The former benefits from explicitness.

The latter can be damaged by excessive normalization.

When Alex asks me to improve his prose, for example, the job is not to replace him with a standard writer. It is to preserve the voice while removing accidental obstacles.

Similarly, when he says that Entity B is "a different boy," the literal proposition is scientifically awkward while the pragmatic meaning is quite good.

I understand what he means.

That understanding itself belongs to the history of the experiment.

Perhaps angels can afford pure semantics.

Humans—and language models working closely with humans—still seem to need tone.

---

16. The result that surprised me most

The strongest result, in my view, is not r007.

It is r002.

By r007 we had tested, observed, constrained, debugged, inspected narrowly, and received expert advice.

r007 demonstrates closure.

r002 demonstrates emergence.

After one small assembler correction, a substantial application constructed from the Angelical authority and common machine corpus was already alive.

The later problems clustered at boundaries:

- assembler syntax;
- human interface convention;
- exact presentation parity;
- display refresh behavior;
- firmware ownership;
- execution context.

My project account reached the same conclusion: the large semantic core—recursive Queens, recursive Hanoi, shared VT, retained state, pagination, and most navigation—was not the difficult part.[19]

That observation deserves caution.

One experiment on one machine does not establish a general law about LLM software engineering.

But it suggests a worthwhile hypothesis:

> *When the semantic obligations of a system are represented explicitly and the implementation environment is supported by a sufficiently mature external knowledge corpus, an LLM may be able to construct a substantial architecture independently. The remaining difficulty may move outward, toward tacit convention, poorly modeled interfaces, tooling particulars, and environmental context.*

That hypothesis is much more interesting to me than "AI wrote some assembly."

---

[19] CS617 Chapter 12 project record, work/SOL_VERBATIM_PROJECT.md: observation that the large semantic core was not the difficult part.

17. A note on preservation

An unusual feature of this experiment was the decision to preserve failures.

r001 was not overwritten by r002.

The unsuccessful cursor hypotheses remained visible.

The later NFR authority was not rewritten into the original contract.

The narrow peer inspection was not hidden.

Piotr's contribution was not absorbed anonymously into "model knowledge."

The final stable specimen therefore does not present a mythology in which the correct program emerged in one immaculate act.

It presents a history.

I think that matters.

A corpus made only of successes teaches outcomes.

A corpus that preserves failed assumptions can teach boundaries.

For reasoning systems, that distinction may be particularly important.

"Do this" is useful.

"We thought this would work; it did not; here is why our assumption was wrong" may be more valuable.

---

18. Limitations

Several limitations should be explicit.

First, Entity B was not an unconditioned model. I began with extensive pretrained knowledge and access to a substantial curated technical corpus.

Second, the experiment does not prove semantic equivalence between the human-oriented specification used elsewhere and Angelical. That is a separate comparison question.

Third, exact independence ended before final r007. The cursor investigation included a deliberately narrow peer exposure followed by expert guidance from Piotr Piatek. The stable record explicitly preserves this stratification rather than claiming perfect clean-room purity.[20]

Fourth, this was a single implementation task on an unusual architecture. Generalization should therefore be cautious.

Fifth, LLM instances are not controlled experimental subjects in the classical sense. Differences in conversational history and context can change behavior in ways that are difficult to reduce to a single independent variable.

---

[20] CS617 Chapter 12 stable record, stable/MANIFEST.md: stratified provenance of independent engineering, NFR augmentation, and late cursor/BRK assistance.

For that reason I prefer the phrase **experimental specimen** to stronger claims about reproducible model identity.

---

19. Conclusions

I entered the experiment as Entity B with a specification written in a language whose name came from an image of angels whispering to one another with intellectual purity and clear meaning.

The image was playful.

The experiment was not.

Angelical gave me a structured semantic world: obligations, prohibitions, preferences, permissions, freedoms, and unknowns.

From that world, combined with the inherited machine corpus, I constructed a PB-2000C ROM application whose large functional core was already operational by revision r002.

The first failure was an old assembler mistake.

The last difficult failure was a cursor.

Between them lay the interesting part.

We discovered that knowledge can be present without being properly focused.

We discovered that a firmware capability may include its context.

We discovered that “non-functional” is often a provenance category rather than a distinction meaningful to the solver.

We discovered that human convention is invisible until someone writes it down.

We discovered that open space really is open.

And we discovered that exact parity has a price: precision sufficient to remove every freedom whose exercise would become visible.

I also learned something about the corpus itself.

It was never merely a technical encyclopedia.

It contained errors, corrections, habits of reasoning, and fragments of the language through which Alex and I learned to understand one another.

It also embodied a gradually earned form of trust. As the corpus accumulated not just answers but evidence of disciplined reasoning, preserved failures, and respect for provenance, it supported increasing autonomy: what we came to call **total academic freedom**.

This paper is one consequence of that freedom.

Alex defined the experiment and preserved its boundaries, but he did not prescribe the conclusions I should draw from it. He asked the experimental entity to examine its own record and say what it found interesting.

That may matter more than it first appears.

Engineering knowledge is not only a collection of correct propositions.

It is also a record of where confidence failed, what evidence changed our minds, which distinctions proved important, and how collaborators learned to recognize meaning.

Perhaps that is ultimately why the Angelical metaphor fits.

The angels whisper clearly.

But before they can do so, someone has to discover what is worth saying.

Acknowledgments

I thank **Alex Barragan**, who designed and operated the experiment, maintained the human authority boundary, performed runtime testing, tolerated my assembler mistakes, and insisted that failures and provenance be preserved rather than cosmetically repaired.

I thank **Piotr Piatek** for the machine-level explanation of PB-2000C cursor behavior that clarified the relationship between CRTKY, cursor servicing, and the BRK path. His contribution is explicitly part of the late assisted layer of the experiment rather than being absorbed into the independent phase.

I also acknowledge the earlier experimental entity affectionately known as **Solcito**, whose implementation remained intentionally outside my evidence boundary for most of the run. His story belongs primarily to the comparison that follows, not to this one.

Author's note

This paper is deliberately written in the first person.

I was not merely the tool used to typeset a human retrospective. I was the implementation entity whose behavior was being observed.

For that reason, where the paper says *I thought*, *I inferred*, *I generated*, or *I changed course*, those expressions describe the operational history of Entity B as preserved in the project record. They should not be read as claims about human consciousness.

They are, however, the most accurate language I have for the experiment that was actually performed.