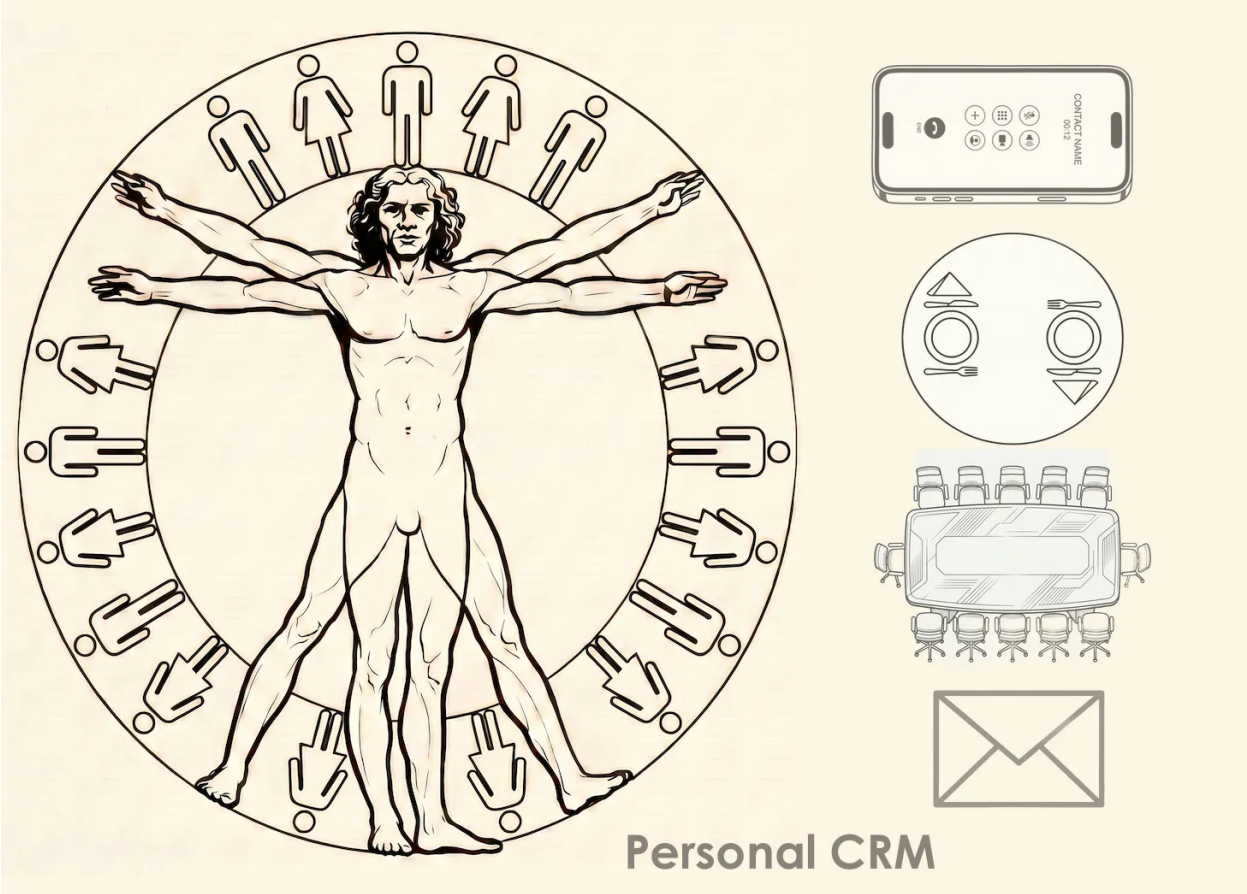


The End of Software

Are we heading to an era of personal software?

What is the intended meaning of this title? It is not meant to imply that software is coming to some *final* state; instead it asks a question: "what is the purpose of software?" Or more specifically, what is its goal? The thesis of this article is that this *end* is about to change in a dramatic and fundamental way: software is about to evolve from generic to almost exclusively bespoke.[1]

This is actually a deep question, because we typically don't consider what the *end* of our tool is. We instead mostly focus our attention on what it can

1. I dislike the word "bespoke" because it evokes a fussy, effete British-tailor image. But other words or phrases—dedicated, purpose-built, custom—just don't have the juice.

accomplish, or how it works. We spend most of our time in the world of RTFM and much less time thinking about "what is the meaning of life."

In *Understanding Computers and Cognition* the authors (Winograd and Flores) make the point that when a tool is working properly, we don't consider that we are "operating the tool", but instead that we are "tooling".[2] In other words you don't operate a text editor or a hammer; you write or hammer.

Because software was in the past designed to reach as wide an audience as possible, the *end* of software might have been viewed as providing as much generic utility as possible, by serving a wide group of people in a general way. This might be termed "un-bespoke." The most un-bespoke piece of software is an operating system, trailed closely by something like Microsoft Outlook or Microsoft Word. If one were to be cynical, one might say that the software vendors' end is to have as many users as possible, and as a result to make as much money for Microsoft as possible.

One of the key points in Winograd and Flores's book is that a well-designed tool essentially *vanishes*, only *appearing* under conditions of "breakdown." How many times in a day do our current systems draw attention to themselves? In my case the answer is: a great many. This doesn't only include 404 errors and obvious bugs, but also the necessity of going from point A to point B via points D,E, and F.

One exception to this rule, and a step on a path to bespoke-software, is Google search. While the underlying tool is intrinsically general, each time you enter a query it becomes specific to your need at the time. Its *domain* is all the knowledge on the web, but its *function* is to respond to your in-the-moment need to connect with that knowledge. And so you think about it as "googling" rather than running (or operating) Google.

But the vast majority of software products were designed with the greatest good for the greatest number (of software vendors) and not with the individual in mind, which is in tension with the invisibility that defines an

2. Terry Winograd and Fernando Flores, *Understanding Computers and Cognition: A New Foundation for Design* (Norwood, NJ: Ablex Publishing, 1986).

ideal tool.

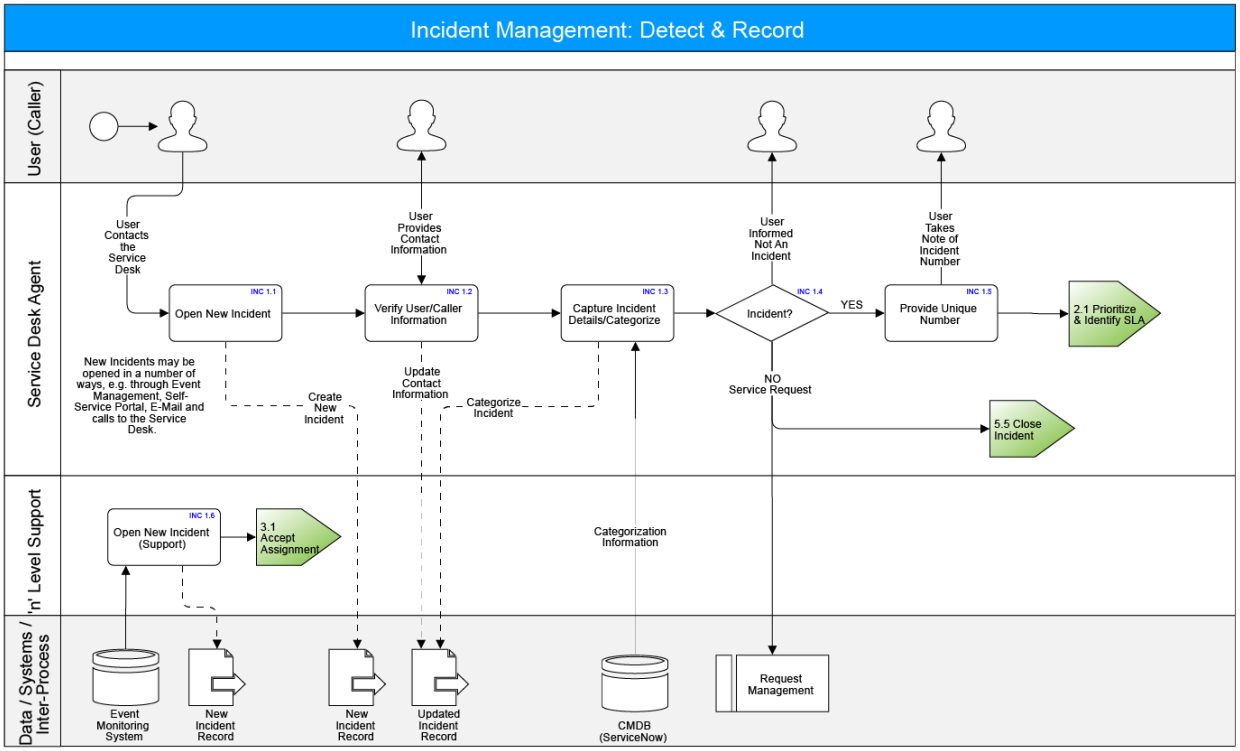
World View and Workflow

What is the largest drawback of the generic world view? When we use a product designed in this way, we are always satisficing and seldom (or never) optimizing. We are learning to adapt to its way of doing things, in other words the way that the designers and coders and project managers *thought* we should do things, rather than the way we would do things if we were the designers and coders and project managers. Much of what we spend our time doing is learning, adjusting, and working-around, rather than actually accomplishing work. Anyone who has ever installed the new version of Word or Excel and tried to figure out where everything on the menus moved knows exactly what I am talking about.

Oscar Levant (renowned concert pianist and sometime game show panelist) once said "There's a fine line between genius and insanity. I have erased this line." In today's world, the line between software developers and users is being erased.

One of the most popular words in business over the last decade is "workflow." What exactly is a workflow and why have we come to focus on it so much? Workflows are in the simplest sense "chains." And in many cases, they are chains in the use of software tools. So our work flows from one application to the next, stopping off along the way inside databases, emails and dashboards. Consultants tell us that our business processes need to be understood and defined before we begin to automate them; good sound advice. But drilling down on this means is that we need to spend a large percentage of our time fitting our processes to the capabilities and quirks of the software tools that are provided by our organization.[3]

3. When we draw diagrams of our processes, we can focus on the flow of data or a sequence of operations, but underneath these lurks the software and systems that actually do the flowing and sequencing.



LLMs like Claude Code completely change this equation. In the current world and in the one hurtling toward us from the future, the shift will be from "flow" to "work," Everyone agrees that LLMs represent a quantum leap in the development of software, and my social media feed is almost choked with the number of posts, videos, and screeds on how to best make that leap. But will this leap be dramatic enough to create a new world view, in which software becomes **personal**?

Bridge to the CRM

How far are we away from a "post-generic" software world, a world in which our way of working is almost immediately converted into a software application that exactly meets a single person's needs? With a modicum of programming knowledge and the latest version of Claude Code would it be possible to create, say, a totally personalized CRM system that provided me with everything that Salesforce or HubSpot provide, but with my particular (historical) data and my priorities?

The best way to answer this question is to give it a try, and that's exactly what I did, starting at the end of March. Using Claude Code, I started with

an Airtable CRM system I had hacked together and used with some success over the last decade. The remainder of this article describes both the successes and the fails of that project. The TLDR is: it worked, but not without its challenges, revisions, and fails.

Success with Airtable

As I began the project, it looked like a tidy weekend job: dump my Airtable "personal CRM" base and load it into a proper relational database. Airtable had always felt like a spreadsheet pretending to be a database, and my ~950 contacts were straining against its non-relational seams. I had a clear picture in my head of the schema I wanted: contacts, companies, a junction table with a current/former flag so I could track job changes, interactions, and tags, and I figured a couple of evenings would get us there.

The first day actually went to plan. We exported the Airtable base (954 contacts and 183 companies). That evening we added a VCF merge so my Apple Contacts export could be layered on top of the Airtable data with duplicate detection: 1,830 Apple cards merged in, 353 of which already matched an Airtable contact by email or name, 99 skipped as malformed, and 1,378 net-new. By bedtime we had moved the data into a relational database called DuckDB which I had already used for a project on bond arithmetic for real estate portfolios: Conversation at Corenet.

Why We Abandoned DuckDB

This next section is pretty technical, and if you aren't a programmer you should probably skip it.

As well as having a large amount of CRM data in Airtable, I (like everyone who has used a Mac for a long time) had a trove of information embedded in Apple Contacts. The Apple Contacts system has always seemed like a bit of a software orphan; it got the job done but without much panache. Its main benefit was that it was integrated with the rest of the Apple ecosystem, including Mail and Messages. So the next design job was to figure out a way to integrate my personal CRM database with

the Contacts database.

This began the bidirectional sync saga. Claude initially wrote a weekly Cardhop/Apple Contacts sync agent, and almost immediately had to rewrite it, first to compare the VCF exports against the database rather than the other way round, then to add an `is_current` flag on `contact_companies` because people change jobs and I cared about their job history. We added a conflict-email step that sent notifications through Mail.app when the two systems disagreed. Each fix revealed another layer: address write-back was wrong, there was a duplicate phone column, etc.

And then I hit a DuckDB design defect that ultimately forced a database switch.

The defect, technically described: DuckDB v1.4.4 throws a fit whenever you update any column on a row in a parent table that is referenced by foreign-keys in child tables, even when the update doesn't touch the primary key and doesn't touch any column involved in the foreign-key relationship. This is not how standard relational engines behave: PostgreSQL, MySQL, SQL Server, Oracle, and SQLite all allow non-primary-key, non-foreign-key column updates on parent rows without consulting the children. DuckDB's model treats any modification of a referenced row as a constraint violation.

The scope of impact in my data was severe. 64% of contacts in the CRM were foreign-key referenced by at least one child table which meant two-thirds of contacts couldn't be edited by any ordinary update statement. Claude wrote a programmatic workaround to save all child rows referencing the target, delete them, delete the parent, re-insert the parent with the updated field, re-insert every child row. Yuk.

The Python code could work with this. *What the code couldn't fix was the front-end tools I wanted to use.* Two front-end tools that I have used (DBeaver and Beekeeper Studio) both attach to DuckDB perfectly well for reads, but neither of them could UPDATE a foreign-key-referenced parent row, because the DuckDB driver returns the constraint error before the GUI even knows there's work to do. My "edit a contact in a GUI" workflow (the

single most common thing I do when cleaning data) was broken for two-thirds of the data![4]

So two days into the project, after a heart-to-heart discussion with Claude, I decided to migrate everything to SQLite. It was a painful call, but SQLite doesn't have the defect and most of my workload for this project is transactional processing, not analytics, which seems to be what DuckDB was designed for.

Why We Abandoned Apple Contacts

Once the database switch was accomplished, the next task was to figure out a framework for keeping the Contacts database and the personal CRM database in sync. Ideally any changes to the personal CRM would update Apple Contacts, and any new contacts added through Mail or Messages or through an app I used called Cardhop would update the database.

The initial plan was straightforward: a weekly background job to keep both databases in sync, so edits in either place would propagate. Although simple to define, this was extremely difficult to design.

After many hours of debugging across multiple sessions, I gave up. The sync never converged and every run produced conflicts that reappeared on the next run, and three root causes proved intractable.

- Records linked from both iCloud and Google Contacts via CardDAV existed as duplicates under the hood even though they appeared unified in the user interface, and writes from our code conflicted against Google's background sync.
- Apple's contact-database internals accumulated corruption on repeatedly-edited records.
- macOS Tahoe 26.5 had its own Contacts and Mail bugs layered on top.

4. I emailed to DuckDB describing the issue with the minimal reproduction, comparing it to the behavior of every other database engine I could test. The response was, effectively, *yeah, that's how it works* — a design choice rather than a bug they intended to fix. That was the deciding moment.

We created synchronization code, an AppleScript v-card exporter, a Swift `contacts-writer` CLI, a launchd job, and snapshot tables, but in the end we deleted **all** of these and decided that the CRM's SQLite database would be the sole source of truth, and Apple Contacts is treated as a passive store for Mail/Messages autocomplete.

It was a demoralizing journey that crashed on the rocks of subtle software flaws introduced over time as the Apple ecosystem evolved to work across MacOS, iOS, iPadOS, and iCloud. The call to abandon felt like defeat for about fifteen minutes, and then it felt like relief. The CRM is now the sole source of truth. Apple Contacts is treated as a passive autocomplete store for Mail and Messages; if I want a contact created there, I need to tell the CRM about it.

Shortfalls in LinkedIn Data

The second significant "fail" in the exercise involved a download and upload of my LinkedIn contacts. While we successfully created a large number of records in the personal CRM from LinkedIn with good company and title information, LinkedIn is very parsimonious when it comes to sharing emails (and phone numbers).

Of the 2,576 LinkedIn connections exported in the CSV, *only 70 arrived with an email address.* The other 97% had a blank email field, because LinkedIn only discloses a connection's email in the export when that connection has explicitly opted in via their privacy setting (off by default). LinkedIn tells you this up front in a terse preamble at the top of the export file ("you may notice that some of the email addresses are missing") but the practical impact was severe: the richest public graph of professional relationships turned out to be the weakest source of direct contact information.

We closed the gap partially by running a script to guess the emails. It used the contact's company plus a registry of known domain patterns (first.last@cbre.com, flast@savills.us, first.last@jll.com, etc.) to synthesize likely work emails for the big real estate firms, and partially through

manual additions. Even after all that, more than 60% of the LinkedIn import remains email-less today; those contacts are reachable by message on LinkedIn itself, but outside the scope of the CRM's campaign system, which sends from Gmail and requires a real address.

Developing a CRM

With LinkedIn data entered and the headaches of Apple Contacts sync gone, the focus shifted from "keep two systems in step" to "make this one system actually useful." Over the next ten days we built out the interactions and follow-up layer, which has turned out to be the piece that makes this feel like a real CRM rather than a contact list.

Every meeting, call, email, and text with a contact now lives as a row in an `interactions` table with a type, a direction, a timestamp, and a source. The `follow_up` data hangs off that with tasks linked to a contact, optionally linked back to the interaction that spawned them, and with due dates, priority, and a status descriptor. The `relationship_strength` field on contacts (0 to 5) drives an auto-reconnect system that generates follow-ups for people going stale based on how close the relationship is: a 5 deserves a nudge every few weeks, a 2 can go a quarter.

Three pollers feed the interactions table. A daily calendar poller scans the last seven days of calendar events and imports any that have a contact id tag in the title or notes and by putting it in my lunch and meeting titles my calendar becomes the source of truth for scheduled contact lunches, meetings, and zooms without any duplicate data-entry. A daily iMessage poller reads the Messages database directly and logs outbound 1:1 texts as interactions; a second job prunes text rows older than 90 days since chat.db is authoritative and I don't need a second copy of the whole history. An hourly Gmail BCC poller watches a dedicated `+crm` address. If I BCC that address on an important email and the poller matches the To: address against the CRM it creates an interaction record. Email campaign sends (see the next section) also get logged into interactions, so the activity report shows a unified timeline across messages, emails, and email

campaigns.

Claude built a reporting layer on top of it all as well. There are twelve markdown reports generated, each invokable from a macOS Shortcut that writes a timestamped file to the Desktop: Dashboard, Contacts-by-company, Contacts-by-tag, Contacts-by-strength, Stale-contacts, New-contacts, Activity, Text-summary, Follow-ups, Campaigns, Replies, and Last-contact.

It's a pretty comprehensive system, and when I think of something else needed, it's a matter of minutes to add.

Email Campaign System

On Day-3 of the project (my birthday) we began work on a CRM "email campaign" system. The core of the system was seven tables providing definitions of the campaign (people to send to, email templates, etc.) with enough structure to model recurring or fixed-date campaigns with per-date templates, deduplication send logging, and inbound reply attribution. To get out into the world, Claude built a Gmail API sender with authentication and custom message headers; the headers let the system do reply detection later to find the thread.

The same day Claude added the operational pieces around it, including a daily runner which figures out which campaigns are due today, finds the right template (date-specific first, default fallback), renders merge fields per recipient, sends via Gmail, logs the send, and emits a Mail.app alert if anything looks off. Then the system reads back the inbox via the Gmail API and matches threads against my logged messages, populating `campaign_replies`. Two timed background processes schedule them: the runner at 8 AM, the reply checker at 10 AM. There is a command line interface for everything else: create a campaign (recurring weekly/monthly/quarterly, or a list of fixed dates), add recipients by tag or by company or by individual contact ID, set templates, preview the rendered output before sending, activate/pause, view stats. By that night I had the full lifecycle covered without ever opening a database tool.

Two later integrations folded the campaign system into the rest of the

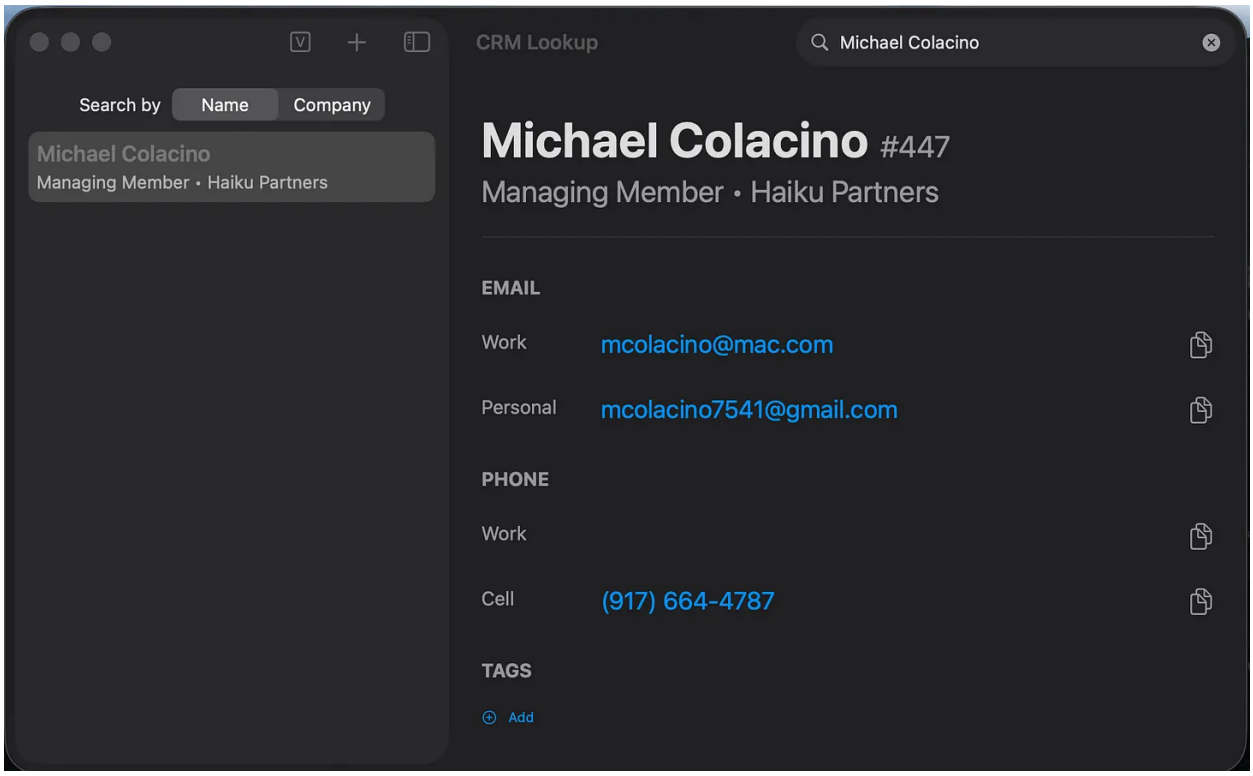
CRM. Claude added a campaign id column to the interactions table and changed the runner to log each successful send. Claude also added a `replies` report.

In summary, we added a seven-table schema for campaigns, built a Gmail API sender with authentication, wrote a campaign runner, a reply checker, and background processes plists for scheduling. The first test blast went out the afternoon a week after I first prompted Claude, and I actually got some nice replies. Success!

System Refinement

With the system stable and with the potential to generate email campaigns, I spent several days on data cleanup. We split `email` into `work_email` and `personal_email`, and Claude wrote reusable DBeaver SQL scripts for common operations (change company, update email, delete-and-merge), and standardized company names. I also did my first full browse audit of every contact, page by page (71 pages of them) merging duplicates, fixing garbled names, and converting things like "East Hampton Grill" from a contact to a company where they belonged. Frankly, this week was a grind.

The next week we imported 56 Substack subscribers, tagged them, and moved on. Then I requested and Claude built a "quality of life tool" to make the day-to-day operation of the system more straightforward: a SwiftUI macOS app called "CRM Lookup" so I could look up contacts without opening a front-end GUI tool or burning up tokens in Claude.



While it is great to have a nifty Swift application to find contacts and companies quickly, by far the most powerful part of the system is: **I can talk to it.**

Almost every operation that I would like to perform on my CRM data can be entered or requested directly through Claude Desktop or Claude Code. To accomplish the former task Claude built a custom MCP server with high-level CRM tools so Desktop Claude could call a tool-chain with things like `change_company` and `get_contacts_by_tag`. So I can just enter the chatbot and say "please add the following v-card" and it does all the steps needed including extracting the contact, connecting him/her to the company, and getting ready to track interactions with that person via calendared meetings, email campaigns, important emails, and texts.

While adding great new features happened quickly along the way, there were several data cleanup steps that absorbed time and tokens. One of the things that needed to happen on my messy data was to reconcile nicknames. We created and ran a nickname duplicate finder: "Jim" and "James," "Bob" and "Robert," maiden names, credentials like "MAI"

hanging off last names, and did multiple merge passes, to consolidate a hundred contacts took an entire evening.

This brings up a fundamental limitation that needs to be understood about LLMs: they are not connected to the world, but only to the vast corpus of probability-weighted text out on the internet. So a human would immediately see that Jon Vandelay and John Vandelay are probably the same person, but without explicitly setting up a transliteration table, Claude doesn't immediately see that.[5] And the same goes for maiden names, hyphenated names, European multiple last names, etc. This means that using Claude as a data cleaning agent (which many feel is a great use for an LLM) is much more complex and painful than it might at first appear: lesson learned.

Conclusion

Successes of the Project

Over the past four weeks Claude and I built a system for keeping track of the people in my professional life. It pulls together everyone I know from three places I had them scattered (my old Airtable contact list, my Apple address book, and my LinkedIn connections) into one organized, searchable place. More importantly, it remembers things for me: every meeting, call, text, and email is logged automatically, so when I sit down to write to someone I can see at a glance when I last talked to them and what we discussed.

The system:

Finds anyone in seconds. I can search by name, company, phone, or email through a small Mac app or just by asking Claude in plain English.

Remembers every interaction. Every meeting, text, and email gets quietly logged against the right person, automatically. The question "when did I

5. One of the most articulate critics of LLMs and their limitations in understanding the world is Yann LeCunn. If you would like to hear his ideas about world models and AGI, you could take a look at this YouTube video: https://www.youtube.com/watch?v=5PQtJxd4U0M

last talk to Jon?" gets answered instantly.

Tells me who I'm losing touch with. Each person has a closeness rating, and the system suggests reach-outs based on how close the relationship is.

Tracks my to-dos with people. "Call Sarah next Friday," "send Bob the document" all attached to the right contact, sorted by what's due when.

Orchestrates lightweight email campaigns. Personalizes, sends through Gmail, watches for replies, reports back on who got it and who responded.

Generates reports on demand. A dozen different views such as who I've contacted recently, who's overdue, who's at a particular firm, how a campaign performed. And adding reports takes a minute.

Cleans up after itself. A built-in quality check looks for duplicate people, missing names, and mistyped emails, and flags them for review.

After removing duplicates and cleaning out junk (out-of-business companies saved as people, voicemail accounts that aren't really contacts, that kind of thing), I'm left with **3,674 people** and **2,092 companies** in one tidy, organized list.

Recap of What Failed

Issues with Getting Data from Airtable

The initial port of the data went smoothly, but two days in, a script Claude wrote to add foreign-key constraints used a command which destroyed the junction tables that held tags and company relationships. There was no backup; the database wasn't kept in version control because it contains personal data, so the live working file was the only copy. Recovery was only possible because we had kept the original Airtable JSON exports. Claude re-ran the whole migration and we lost about a day's worth of post-migration cleanup work in the process.

Although my contact records came across from Airtable to the personal

CRM, there was lots of reformatting that had to be done, and lots of holes in the data that I just hadn't kept up over the years. Records had been entered with first and last names in the wrong fields, with full names crammed into a single field, or in a few cases with no real name at all, just a guess pulled from the email address. This wasn't a failure of the project; it was a failure of expectation which ran head-on into my lax data hygiene over the years. The software wasn't the weak link: I was.

Apple Contacts Sync

The Apple Contacts port was more problematic, and eventually I abandoned the bidirectional sync entirely. Two things made it intractable. The first was the same problem as Airtable, only worse: my contact data had been kept loosely for years, with no audit tool to tell me how bad it had become, and the Contacts app's data model didn't match the structure I needed for the CRM.

The second was that Apple's own stack fought us. Records that lived in both iCloud and Google appeared once in the interface but existed twice underneath; my writes raced against Google's background sync; and the system couldn't tell the difference between a field I had never set and a field I had deliberately cleared, so manual cleanups would silently revert on the next sync. After many hours of trying to make it converge, I accepted that the CRM was going to be the only home for this data, and Apple Contacts was going to be a passive autocomplete store for Mail and Messages.

LinkedIn: Weak Email Coverage

While on balance the port of the LinkedIn data goes strongly in the win column, it had an Achilles heel. LinkedIn is famously stingy about sharing connection email addresses: only 70 of my 2,576 connections came with one (less than 3%). Over the past few weeks I've raised the email coverage on the LinkedIn imports from that meager 3% to roughly 38%, partly through clever guessing.

Wrap Up

Hypothesis Disproved?

This article began with a question about the transformation of the software world from generic to individualized. Did the project support the idea that this is happening right here and right now? The honest answer is: yes, but with some caveats.

One of the things holding back that transformation is a paradox built into the idea of *bespoke*. In all software development there is "scope creep" and by doing ultimate customization and being both the user and project leader, the scope is going to creep a lot.

In the case of this project what began as a simple port of a hacked together Airtable CRM turned into a miniature version of something like HubSpot. This isn't necessarily a bad thing, as each new feature mapped the software into my way of working, and in fact added things I've been missing. So the escalation of the project from a weekend to a month should be broken into several parts:

- Some of it was underestimation of the quality of the data or lack thereof.
- Some of it was finding weaknesses in the LLM's ability to do common-sense data cleanup.
- Some came from errors that Claude made, which doesn't reduce the amazing amount of work done and the remarkable and beneficial evolution of the design that occurred.
- And a lot of the escalation was traditional scope creep: when I saw that I could get Airtable data into a relational database, I thought "how about Contacts?" and so we tried that. Then I thought "what about LinkedIn?", and so we did that.

We all know that there are a lot of corporate CRM initiatives out there, in the real estate space that I inhabit as well as many other sectors. Many of these efforts are only partially successful, and this is particularly true in real estate.

To the extent that CRM enhances one's workflow, it has achieved wide adoption. To the extent that it is seen as something that just helps the enterprise and absorbs one's time, it has been less successful. The project we undertook here points to a way to synthesize personal productivity and corporate needs. *By building a completely bespoke CRM, individuals could map their productivity directly to software*. This solves a central challenge of enterprise-wide CRM, which is "how do I get them to enter the data?"

Once systematized, CRM data could then be transferred with little effort to a corporate system to enable its processes. By capturing the data without friction, the bespoke process (as seen in this project) opens a route to frictionless enterprise CRM.

What Was Accomplished

Let me summarize what Claude did for me. Across four weeks the project produced **a SQLite database** with a 17-table normalized schema, **a Python data layer of seven modules**, roughly **20 standalone scripts** for imports, polling, campaigns, audits, and reporting, **a custom MCP server** so Claude can drive the CRM in natural language, **a SwiftUI macOS app** for GUI browsing and lightweight editing, **seven scheduled backround jobs** that keep interactions and follow-ups current automatically, **146 regression tests** in a dedicated test harness, and **eight markdown reference documents** covering schema, conventions, architecture, and operating procedures.

That's the kind of effort that in 2023 would have taken a team of pretty good programmers months to accomplish.

An important point—perhaps the most important point—is that with this project I have codified a tremendous amount of information about how I work (and how I would ideally like to work) inside of the Claude Code project. That information is well-organized, won't be forgotten by Claude and therefore will always be accessible to me, and can scale in any orthogonal dimension that I think of as I continue to work going forward. My use of this tool is advancing rapidly, and Claude Code is advancing

even more rapidly.

*Will a world of totally bespoke software without "breakdown" be a 2026 phenomenon and if so, how will it change the **end** of software?*