What I have worked on

February 2021

Before college the two main things I worked on, outside of school, were writing and programming. I didn't write essays. I wrote what beginning writers were supposed to write then, and probably still are: short stories. My stories were awful. They had hardly any plot, just characters with strong feelings, which I imagined made them deep.

The first programs I tried writing were on the IBM 1401 that our school district used for what was then called "data processing." This was in 9th grade, so I was 13 or 14. The school district's 1401 happened to be in the basement of our junior high school, and my friend Rich Draves and I got permission to use it. It was like a mini Bond villain's lair down there, with all these alien-looking machines — CPU, disk drives, printer, card reader — sitting up on a raised floor under bright fluorescent lights.

The language we used was an early version of Fortran. You had to type programs on punch cards, then stack them in the card reader and press a button to load the program into memory and run it. The result would ordinarily be to print something on the spectacularly loud printer.

I was puzzled by the 1401. I couldn't figure out what to do with it. And in retrospect there's not much I could have done with it. The only form of input to programs was data stored on punched cards, and I didn't have any data stored on punched cards. The only other option was to do things that didn't rely on any input, like calculate approximations of pi, but I didn't know enough math to do anything interesting of that type. So I'm not surprised I can't remember any programs I wrote, because they can't have done much. My clearest memory is of the moment I learned it was possible for programs not to terminate, when one of mine didn't. On a machine without time-sharing, this was a social as well as a technical error, as the data center manager's expression made clear.

With microcomputers, everything changed. Now you could have a computer sitting right in front of you, on a desk, that could respond to your keystrokes as it was running instead of just churning through a stack of punch cards and then stopping. [1]

The first of my friends to get a microcomputer built it himself. It was sold as a kit by Heathkit. I remember vividly how impressed and envious I felt watching him sitting in front of it, typing programs right into the computer.

Computers were expensive in those days and it took me years of nagging before I convinced my father to buy one, a TRS-80, in about 1980. The gold standard then was the Apple II, but a TRS-80 was good enough. This was when I really started programming. I wrote simple games, a program to predict how high my model rockets would fly, and a word processor that my father used to write at least one book. There was only room in memory for about 2 pages of text, so he'd write 2 pages at a time and then print them out, but it was a lot better than a typewriter.

Though I liked programming, I didn't plan to study it in college. In college I was going to study philosophy, which sounded much more powerful. It seemed, to my naive high school self, to be the study of the ultimate truths, compared to which the things studied in other fields would be mere domain knowledge. What I discovered when I got to college was that the other fields took up so much of the space of ideas that there wasn't much left for these supposed ultimate truths. All that seemed left for philosophy were edge cases that people in other fields felt could safely be ignored.

I couldn't have put this into words when I was 18. All I knew at the time was that I kept taking philosophy courses and they kept being boring. So I decided to switch to AI.

AI was in the air in the mid 1980s, but there were two things especially that made me want to work on it: a novel by Heinlein called The Moon is a Harsh Mistress, which featured an intelligent computer called Mike, and a PBS documentary that showed Terry Winograd using SHRDLU. I haven't tried rereading The Moon is a Harsh Mistress, so I don't know how well it has aged, but when I read it I was drawn entirely into its world. It seemed only a matter of time before we'd have Mike, and when I saw Winograd using SHRDLU, it seemed like that time would be a few years at most. All you had to do was teach SHRDLU more words.

There weren't any classes in AI at Cornell then, not even graduate classes, so I started trying to teach myself. Which meant learning Lisp, since in those days Lisp was regarded as the language of AI. The commonly used programming languages then were pretty primitive, and programmers' ideas correspondingly so. The default language at Cornell was a Pascal-like language called PL/I, and the situation was similar elsewhere. Learning Lisp expanded my concept of a program so fast that it was years before I started to have a sense of where the new limits were. This was more like it; this was what I had expected college to do. It wasn't happening in a class, like it was supposed to, but that was ok. For the next couple years I was on a roll. I knew what I was going to do.

For my undergraduate thesis, I reverse-engineered SHRDLU. My God did I love working on that program. It was a pleasing bit of code, but what made it even more exciting was my belief — hard to imagine now, but not unique in 1985 — that it was already climbing the lower slopes of intelligence.

I had gotten into a program at Cornell that didn't make you choose a major. You could take whatever classes you liked, and choose whatever you liked to put on your degree. I of course chose "Artificial Intelligence." When I got the actual physical diploma, I was dismayed to find that the quotes had been included, which made them read as scare-quotes. At the time this bothered me, but now it seems amusingly accurate, for reasons I was about to discover.

I applied to 3 grad schools: MIT and Yale, which were renowned for AI at the time, and Harvard, which I'd visited because Rich Draves went there, and was also home to Bill Woods, who'd invented the type of parser I used in my SHRDLU clone. Only Harvard accepted me, so that was where I went.

I don't remember the moment it happened, or if there even was a specific moment, but during the first year of grad school I realized that AI, as practiced at the time, was a hoax. By which I mean the sort of AI in which a program that's told "the dog is sitting on the chair" translates this into some formal representation and adds it to the list of things it knows.

What these programs really showed was that there's a subset of natural language that's a formal language. But a very proper subset. It was clear that there was an

unbridgeable gap between what they could do and actually understanding natural language. It was not, in fact, simply a matter of teaching SHRDLU more words. That whole way of doing AI, with explicit data structures representing concepts, was not going to work. Its brokenness did, as so often happens, generate a lot of opportunities to write papers about various band-aids that could be applied to it, but it was never going to get us Mike.

So I looked around to see what I could salvage from the wreckage of my plans, and there was Lisp. I knew from experience that Lisp was interesting for its own sake and not just for its association with AI, even though that was the main reason people cared about it at the time. So I decided to focus on Lisp. In fact, I decided to write a book about Lisp hacking. It's scary to think how little I knew about Lisp hacking when I started writing that book. But there's nothing like writing a book about something to help you learn it. The book, On Lisp, wasn't published till 1993, but I wrote much of it in grad school.

Computer Science is an uneasy alliance between two halves, theory and systems. The theory people prove things, and the systems people build things. I wanted to build things. I had plenty of respect for theory — indeed, a sneaking suspicion that it was the more admirable of the two halves — but building things seemed so much more exciting.

The problem with systems work, though, was that it didn't last. Any program you wrote today, no matter how good, would be obsolete in a couple decades at best. People might mention your software in footnotes, but no one would actually use it. And indeed, it would seem very feeble work. Only people with a sense of the history of the field would even realize that, in its time, it had been good.

There were some surplus Xerox Dandelions floating around the computer lab at one point. Anyone who wanted one to play around with could have one. I was briefly tempted, but they were so slow by present standards; what was the point? No one else wanted one either, so off they went. That was what happened to systems work.

I wanted not just to build things, but to build things that would last.

In this dissatisfied state I went in 1988 to visit Rich Draves at CMU, where he was in grad school. One day I went to visit the Carnegie Institute, where I'd spent a lot of time as a kid. While looking at a painting there I realized something that might seem obvious, but was a big surprise to me. There, right on the wall, was something you could make that would last. Paintings didn't become obsolete. Some of the best ones were hundreds of years old.

And moreover this was something you could make a living doing. Not as easily as you could by writing software, of course, but I thought if you were really industrious and lived really cheaply, it had to be possible to make enough to survive. And as an artist you could be truly independent. You wouldn't have a boss, or even need to get research funding.

I had always liked looking at paintings. Could I make them? I had no idea. I'd never imagined it was even possible. I knew intellectually that people made art — that it didn't just appear spontaneously — but it was as if the people who made it were a different species. They either lived long ago or were mysterious geniuses doing strange things in profiles in Life magazine. The idea of actually being able to make art, to put that verb before that noun, seemed almost miraculous.

That fall I started taking art classes at Harvard. Grad students could take classes in any department, and my advisor, Tom Cheatham, was very easy going. If he even knew about the strange classes I was taking, he never said anything.

So now I was in a PhD program in computer science, yet planning to be an artist, yet also genuinely in love with Lisp hacking and working away at On Lisp. In other words, like many a grad student, I was working energetically on multiple projects that were not my thesis.

I didn't see a way out of this situation. I didn't want to drop out of grad school, but how else was I going to get out? I remember when my friend Robert Morris got kicked out of Cornell for writing the internet worm of 1988, I was envious that he'd found such a spectacular way to get out of grad school.

Then one day in April 1990 a crack appeared in the wall. I ran into professor Cheatham and he asked if I was far enough along to graduate that June. I didn't have a word of my dissertation written, but in what must have been the quickest bit of thinking in my life, I decided to take a shot at writing one in the 5 weeks or so that remained before the deadline, reusing parts of On Lisp where I could, and I was able to respond, with no perceptible delay "Yes, I think so. I'll give you something to read in a few days."

I picked applications of continuations as the topic. In retrospect I should have written about macros and embedded languages. There's a whole world there that's barely been explored. But all I wanted was to get out of grad school, and my rapidly written dissertation sufficed, just barely.

Meanwhile I was applying to art schools. I applied to two: RISD in the US, and the Accademia di Belli Arti in Florence, which, because it was the oldest art school, I imagined would be good. RISD accepted me, and I never heard back from the Accademia, so off to Providence I went.

I'd applied for the BFA program at RISD, which meant in effect that I had to go to college again. This was not as strange as it sounds, because I was only 25, and art schools are full of people of different ages. RISD counted me as a transfer sophomore and said I had to do the foundation that summer. The foundation means the classes that everyone has to take in fundamental subjects like drawing, color, and design.

Toward the end of the summer I got a big surprise: a letter from the Accademia, which had been delayed because they'd sent it to Cambridge England instead of Cambridge Massachusetts, inviting me to take the entrance exam in Florence that fall. This was now only weeks away. My nice landlady let me leave my stuff in her attic. I had some money saved from consulting work I'd done in grad school; there was probably enough to last a year if I lived cheaply. Now all I had to do was learn Italian.

Only stranieri (foreigners) had to take this entrance exam. In retrospect it may well have been a way of excluding them, because there were so many stranieri attracted by the idea of studying art in Florence that the Italian students would otherwise have been outnumbered. I was in decent shape at painting and drawing from the RISD

foundation that summer, but I still don't know how I managed to pass the written exam. I remember that I answered the essay question by writing about Cezanne, and that I cranked up the intellectual level as high as I could to make the most of my limited vocabulary. [2]

I'm only up to age 25 and already there are such conspicuous patterns. Here I was, yet again about to attend some august institution in the hopes of learning about some prestigious subject, and yet again about to be disappointed. The students and faculty in the painting department at the Accademia were the nicest people you could imagine, but they had long since arrived at an arrangement whereby the students wouldn't require the faculty to teach anything, and in return the faculty wouldn't require the students to learn anything. And at the same time all involved would adhere outwardly to the conventions of a 19th century atelier. We actually had one of those little stoves, fed with kindling, that you see in 19th century studio paintings, and a nude model sitting as close to it as possible without getting burned. Except hardly anyone else painted her besides me. The rest of the students spent their time chatting or occasionally trying to imitate things they'd seen in American art magazines.

Our model turned out to live just down the street from me. She made a living from a combination of modelling and making fakes for a local antique dealer. She'd copy an obscure old painting out of a book, and then he'd take the copy and maltreat it to make it look old. [3]

While I was a student at the Accademia I started painting still lives in my bedroom at night. These paintings were tiny, because the room was, and because I painted them on leftover scraps of canvas, which was all I could afford at the time. Painting still lives is different from painting people, because the subject, as its name suggests, can't move. People can't sit for more than about 15 minutes at a time, and when they do they don't sit very still. So the traditional m.o. for painting people is to know how to paint a generic person, which you then modify to match the specific person you're painting. Whereas a still life you can, if you want, copy pixel by pixel from what you're seeing. You don't want to stop there, of course, or you get merely photographic accuracy, and what makes a still life interesting is that it's been through a head. You want to emphasize the visual cues that tell you, for example, that the reason the color changes suddenly at a certain point is that it's the edge of an object. By subtly emphasizing such things you can make paintings that are more realistic than photographs not just in some metaphorical sense, but in the strict information-theoretic sense. [4]

I liked painting still lives because I was curious about what I was seeing. In everyday life, we aren't consciously aware of much we're seeing. Most visual perception is handled by low-level processes that merely tell your brain "that's a water droplet" without telling you details like where the lightest and darkest points are, or "that's a bush" without telling you the shape and position of every leaf. This is a feature of brains, not a bug. In everyday life it would be distracting to notice every leaf on every bush. But when you have to paint something, you have to look more closely, and when you do there's a lot to see. You can still be noticing new things after days of trying to paint something people usually take for granted, just as you can after days of trying to write an essay about something people usually take for granted.

This is not the only way to paint. I'm not 100% sure it's even a good way to paint. But it seemed a good enough bet to be worth trying.

Our teacher, professor Ulivi, was a nice guy. He could see I worked hard, and gave me a good grade, which he wrote down in a sort of passport each student had. But the Accademia wasn't teaching me anything except Italian, and my money was running out, so at the end of the first year I went back to the US.

I wanted to go back to RISD, but I was now broke and RISD was very expensive, so I decided to get a job for a year and then return to RISD the next fall. I got one at a company called Interleaf, which made software for creating documents. You mean like Microsoft Word? Exactly. That was how I learned that low end software tends to eat high end software. But Interleaf still had a few years to live yet. [5]

Interleaf had done something pretty bold. Inspired by Emacs, they'd added a scripting language, and even made the scripting language a dialect of Lisp. Now they wanted a Lisp hacker to write things in it. This was the closest thing I've had to a normal job, and I hereby apologize to my boss and coworkers, because I was a bad employee. Their Lisp was the thinnest icing on a giant C cake, and since I didn't know C and didn't want to learn it, I never understood most of the software. Plus I was terribly irresponsible. This was back when a programming job meant showing up every day during certain working hours. That seemed unnatural to me, and on this point the rest of the world is coming around to my way of thinking, but at the time it caused a lot of friction. Toward the end of the year I spent much of my time surreptitiously working on On Lisp, which I had by this time gotten a contract to publish.

The good part was that I got paid huge amounts of money, especially by art student standards. In Florence, after paying my part of the rent, my budget for everything else had been $7 a day. Now I was getting paid more than 4 times that every hour, even when I was just sitting in a meeting. By living cheaply I not only managed to save enough to go back to RISD, but also paid off my college loans.

I learned some useful things at Interleaf, though they were mostly about what not to do. I learned that it's better for technology companies to be run by product people than sales people (though sales is a real skill and people who are good at it are really good at it), that it leads to bugs when code is edited by too many people, that cheap office space is no bargain if it's depressing, that planned meetings are inferior to corridor conversations, that big, bureaucratic customers are a dangerous source of money, and that there's not much overlap between conventional office hours and the optimal time for hacking, or conventional offices and the optimal place for it.

But the most important thing I learned, and which I used in both Viaweb and Y Combinator, is that the low end eats the high end: that it's good to be the "entry level" option, even though that will be less prestigious, because if you're not, someone else will be, and will squash you against the ceiling. Which in turn means that prestige is a danger sign.

When I left to go back to RISD the next fall, I arranged to do freelance work for the group that did projects for customers, and this was how I survived for the next several years. When I came back to visit for a project later on, someone told me about a new thing called HTML, which was, as he described it, a derivative of SGML. Markup language enthusiasts were an occupational hazard at Interleaf and I ignored him, but this HTML thing later became a big part of my life.

In the fall of 1992 I moved back to Providence to continue at RISD. The foundation had merely been intro stuff, and the Accademia had been a (very civilized) joke. Now I was going to see what real art school was like. But alas it was more like the Accademia than not. Better organized, certainly, and a lot more expensive, but it was now becoming clear that art school did not bear the same relationship to art that medical school bore to medicine. At least not the painting department. The textile department, which my next door neighbor belonged to, seemed to be pretty rigorous. No doubt illustration and architecture were too. But painting was post-rigorous.

Painting students were supposed to express themselves, which to the more worldly ones meant to try to cook up some sort of distinctive signature style.

A signature style is the visual equivalent of what in show business is known as a "schtick": something that immediately identifies the work as yours and no one else's. For example, when you see a painting that looks like a certain kind of cartoon, you know it's by Roy Lichtenstein. So if you see a big painting of this type hanging in the apartment of a hedge fund manager, you know he paid millions of dollars for it. That's not always why artists have a signature style, but it's usually why buyers pay a lot for such work. [6]

There were plenty of earnest students too: kids who "could draw" in high school, and now had come to what was supposed to be the best art school in the country, to learn to draw even better. They tended to be confused and demoralized by what they found at RISD, but they kept going, because painting was what they did. I was not one of the kids who could draw in high school, but at RISD I was definitely closer to their tribe than the tribe of signature style seekers.

I learned a lot in the color class I took at RISD, but otherwise I was basically teaching myself to paint, and I could do that for free. So in 1993 I dropped out. I hung around Providence for a bit, and then my college friend Nancy Parmet did me a big favor. A rent-controlled apartment in a building her mother owned in New York was becoming vacant. Did I want it? It wasn't much more than my current place, and New York was supposed to be where the artists were. So yes, I wanted it! [7]

Asterix comics begin by zooming in on a tiny corner of Roman Gaul that turns out not to be controlled by the Romans. You can do something similar on a map of New York City: if you zoom in on the Upper East Side, there's a tiny corner that's not rich, or at least wasn't in 1993. It's called Yorkville, and that was my new home. Now I was a New York artist — in the strictly technical sense of making paintings and living in New York.

I was nervous about money, because I could sense that Interleaf was on the way down. Freelance Lisp hacking work was very rare, and I didn't want to have to program in another language, which in those days would have meant C++ if I was lucky. So with my unerring nose for financial opportunity, I decided to write another

book on Lisp. This would be a popular book, the sort of book that could be used as a textbook. I imagined myself living frugally off the royalties and spending all my time painting. (The painting on the cover of this book, ANSI Common Lisp, is one that I painted around this time.)

The best thing about New York for me was the presence of Idelle and Julian Weber. Idelle Weber was a painter, one of the early photorealists, and I'd taken her painting class at Harvard. I've never known a teacher more beloved by her students. Large numbers of former students kept in touch with her, including me. After I moved to New York I became her de facto studio assistant.

She liked to paint on big, square canvases, 4 to 5 feet on a side. One day in late 1994 as I was stretching one of these monsters there was something on the radio about a famous fund manager. He wasn't that much older than me, and was super rich. The thought suddenly occurred to me: why don't I become rich? Then I'll be able to work on whatever I want.

Meanwhile I'd been hearing more and more about this new thing called the World Wide Web. Robert Morris showed it to me when I visited him in Cambridge, where he was now in grad school at Harvard. It seemed to me that the web would be a big deal. I'd seen what graphical user interfaces had done for the popularity of microcomputers. It seemed like the web would do the same for the internet.

If I wanted to get rich, here was the next train leaving the station. I was right about that part. What I got wrong was the idea. I decided we should start a company to put art galleries online. I can't honestly say, after reading so many Y Combinator applications, that this was the worst startup idea ever, but it was up there. Art galleries didn't want to be online, and still don't, not the fancy ones. That's not how they sell. I wrote some software to generate web sites for galleries, and Robert wrote some to resize images and set up an http server to serve the pages. Then we tried to sign up galleries. To call this a difficult sale would be an understatement. It was difficult to give away. A few galleries let us make sites for them for free, but none paid us.

Then some online stores started to appear, and I realized that except for the order buttons they were identical to the sites we'd been generating for galleries. This

impressive-sounding thing called an "internet storefront" was something we already knew how to build.

So in the summer of 1995, after I submitted the camera-ready copy of ANSI Common Lisp to the publishers, we started trying to write software to build online stores. At first this was going to be normal desktop software, which in those days meant Windows software. That was an alarming prospect, because neither of us knew how to write Windows software or wanted to learn. We lived in the Unix world. But we decided we'd at least try writing a prototype store builder on Unix. Robert wrote a shopping cart, and I wrote a new site generator for stores — in Lisp, of course.

We were working out of Robert's apartment in Cambridge. His roommate was away for big chunks of time, during which I got to sleep in his room. For some reason there was no bed frame or sheets, just a mattress on the floor. One morning as I was lying on this mattress I had an idea that made me sit up like a capital L. What if we ran the software on the server, and let users control it by clicking on links? Then we'd never have to write anything to run on users' computers. We could generate the sites on the same server we'd serve them from. Users wouldn't need anything more than a browser.

This kind of software, known as a web app, is common now, but at the time it wasn't clear that it was even possible. To find out, we decided to try making a version of our store builder that you could control through the browser. A couple days later, on August 12, we had one that worked. The UI was horrible, but it proved you could build a whole store through the browser, without any client software or typing anything into the command line on the server.

Now we felt like we were really onto something. I had visions of a whole new generation of software working this way. You wouldn't need versions, or ports, or any of that crap. At Interleaf there had been a whole group called Release Engineering that seemed to be at least as big as the group that actually wrote the software. Now you could just update the software right on the server.

We started a new company we called Viaweb, after the fact that our software worked via the web, and we got $10,000 in seed funding from Idelle's husband Julian. In return for that and doing the initial legal work and giving us business advice, we gave

him 10% of the company. Ten years later this deal became the model for Y Combinator's. We knew founders needed something like this, because we'd needed it ourselves.

At this stage I had a negative net worth, because the thousand dollars or so I had in the bank was more than counterbalanced by what I owed the government in taxes. (Had I diligently set aside the proper proportion of the money I'd made consulting for Interleaf? No, I had not.) So although Robert had his graduate student stipend, I needed that seed funding to live on.

We originally hoped to launch in September, but we got more ambitious about the software as we worked on it. Eventually we managed to build a WYSIWYG site builder, in the sense that as you were creating pages, they looked exactly like the static ones that would be generated later, except that instead of leading to static pages, the links all referred to closures stored in a hash table on the server.

It helped to have studied art, because the main goal of an online store builder is to make users look legit, and the key to looking legit is high production values. If you get page layouts and fonts and colors right, you can make a guy running a store out of his bedroom look more legit than a big company.

(If you're curious why my site looks so old-fashioned, it's because it's still made with this software. It may look clunky today, but in 1996 it was the last word in slick.)

In September, Robert rebelled. "We've been working on this for a month," he said, "and it's still not done." This is funny in retrospect, because he would still be working on it almost 3 years later. But I decided it might be prudent to recruit more programmers, and I asked Robert who else in grad school with him was really good. He recommended Trevor Blackwell, which surprised me at first, because at that point I knew Trevor mainly for his plan to reduce everything in his life to a stack of notecards, which he carried around with him. But Rtm was right, as usual. Trevor turned out to be a frighteningly effective hacker.

It was a lot of fun working with Robert and Trevor. They're the two most independent-minded people I know, and in completely different ways. If you could see inside Rtm's brain it would look like a colonial New England church, and if you could see inside Trevor's it would look like the worst excesses of Austrian Rococo.

We opened for business, with 6 stores, in January 1996. It was just as well we waited a few months, because although we worried we were late, we were actually almost fatally early. There was a lot of talk in the press then about ecommerce, but not many people actually wanted online stores. [8]

There were three main parts to the software: the editor, which people used to build sites and which I wrote, the shopping cart, which Robert wrote, and the manager, which kept track of orders and statistics, and which Trevor wrote. In its time, the editor was one of the best general-purpose site builders. I kept the code tight and didn't have to integrate with any other software except Robert's and Trevor's, so it was quite fun to work on. If all I'd had to do was work on this software, the next 3 years would have been the easiest of my life. Unfortunately I had to do a lot more, all of it stuff I was worse at than programming, and the next 3 years were instead the most stressful.

There were a lot of startups making ecommerce software in the second half of the 90s. We were determined to be the Microsoft Word, not the Interleaf. Which meant being easy to use and inexpensive. It was lucky for us that we were poor, because that caused us to make Viaweb even more inexpensive than we realized. We charged $100 a month for a small store and $300 a month for a big one. This low price was a big attraction, and a constant thorn in the sides of competitors, but it wasn't because of some clever insight that we set the price low. We had no idea what businesses paid for things. $300 a month seemed like a lot of money to us.

We did a lot of things right by accident like that. For example, we did what's now called "doing things that don't scale," although at the time we would have described it as "being so lame that we're driven to the most desperate measures to get users." The most common of which was building stores for them. This seemed particularly humiliating, since the whole raison d'etre of our software was that people could use it to make their own stores. But anything to get users.

We learned a lot more about retail than we wanted to know. For example, that if you could only have a small image of a man's shirt (and all images were small then by present standards), it was better to have a closeup of the collar than a picture of the whole shirt. The reason I remember learning this was that it meant I had to rescan about 30 images of men's shirts. My first set of scans were so beautiful too.

Though this felt wrong, it was exactly the right thing to be doing. Building stores for users taught us about retail, and about how it felt to use our software. I was initially both mystified and repelled by "business" and thought we needed a "business person" to be in charge of it, but once we started to get users, I was converted, in much the same way I was converted to fatherhood once I had kids. Whatever users wanted, I was all theirs. Maybe one day we'd have so many users that I couldn't scan their images for them, but in the meantime there was nothing more important to do.

Another thing I didn't get at the time is that growth rate is the ultimate test of a startup. Our growth rate was fine. We had about 70 stores at the end of 1996 and about 500 at the end of 1997. I mistakenly thought the thing that mattered was the absolute number of users. And that is the thing that matters in the sense that that's how much money you're making, and if you're not making enough, you might go out of business. But in the long term the growth rate takes care of the absolute number. If we'd been a startup I was advising at Y Combinator, I would have said: Stop being so stressed out, because you're doing fine. You're growing 7x a year. Just don't hire too many more people and you'll soon be profitable, and then you'll control your own destiny.

Alas I hired lots more people, partly because our investors wanted me to, and partly because that's what startups did during the Internet Bubble. A company with just a handful of employees would have seemed amateurish. So we didn't reach breakeven until about when Yahoo bought us in the summer of 1998. Which in turn meant we were at the mercy of investors for the entire life of the company. And since both we and our investors were noobs at startups, the result was a mess even by startup standards.

It was a huge relief when Yahoo bought us. In principle our Viaweb stock was valuable. It was a share in a business that was profitable and growing rapidly. But it didn't feel very valuable to me; I had no idea how to value a business, but I was all too keenly aware of the near-death experiences we seemed to have every few months. Nor had I changed my grad student lifestyle significantly since we started. So when

Yahoo bought us it felt like going from rags to riches. Since we were going to California, I bought a car, a yellow 1998 VW GTI. I remember thinking that its leather seats alone were by far the most luxurious thing I owned.

The next year, from the summer of 1998 to the summer of 1999, must have been the least productive of my life. I didn't realize it at the time, but I was worn out from the effort and stress of running Viaweb. For a while after I got to California I tried to continue my usual m.o. of programming till 3 in the morning, but fatigue combined with Yahoo's prematurely aged culture and grim cube farm in Santa Clara gradually dragged me down. After a few months it felt disconcertingly like working at Interleaf.

Yahoo had given us a lot of options when they bought us. At the time I thought Yahoo was so overvalued that they'd never be worth anything, but to my astonishment the stock went up 5x in the next year. I hung on till the first chunk of options vested, then in the summer of 1999 I left. It had been so long since I'd painted anything that I'd half forgotten why I was doing this. My brain had been entirely full of software and men's shirts for 4 years. But I had done this to get rich so I could paint, I reminded myself, and now I was rich, so I should go paint.

When I said I was leaving, my boss at Yahoo had a long conversation with me about my plans. I told him all about the kinds of pictures I wanted to paint. At the time I was touched that he took such an interest in me. Now I realize it was because he thought I was lying. My options at that point were worth about $2 million a month. If I was leaving that kind of money on the table, it could only be to go and start some new startup, and if I did, I might take people with me. This was the height of the Internet Bubble, and Yahoo was ground zero of it. My boss was at that moment a billionaire. Leaving then to start a new startup must have seemed to him an insanely, and yet also plausibly, ambitious plan.

But I really was quitting to paint, and I started immediately. There was no time to lose. I'd already burned 4 years getting rich. Now when I talk to founders who are leaving after selling their companies, my advice is always the same: take a vacation. That's what I should have done, just gone off somewhere and done nothing for a month or two, but the idea never occurred to me.

So I tried to paint, but I just didn't seem to have any energy or ambition. Part of the problem was that I didn't know many people in California. I'd compounded this problem by buying a house up in the Santa Cruz Mountains, with a beautiful view but miles from anywhere. I stuck it out for a few more months, then in desperation I went back to New York, where unless you understand about rent control you'll be surprised to hear I still had my apartment, sealed up like a tomb of my old life. Idelle was in New York at least, and there were other people trying to paint there, even though I didn't know any of them.

When I got back to New York I resumed my old life, except now I was rich. It was as weird as it sounds. I resumed all my old patterns, except now there were doors where there hadn't been. Now when I was tired of walking, all I had to do was raise my hand, and (unless it was raining) a taxi would stop to pick me up. Now when I walked past charming little restaurants I could go in and order lunch. It was exciting for a while. Painting started to go better. I experimented with a new kind of still life where I'd paint one painting in the old way, then photograph it and print it, blown up, on canvas, and then use that as the underpainting for a second still life, painted from the same objects (which hopefully hadn't rotted yet).

Meanwhile I looked for an apartment to buy. Now I could actually choose what neighborhood to live in. Where, I asked myself and various real estate agents, is the Cambridge of New York? Aided by occasional visits to actual Cambridge, I gradually realized there wasn't one. Huh.

Around this time, in the spring of 2000, I had an idea. It was clear from our experience with Viaweb that web apps were the future. Why not build a web app for making web apps? Why not let people edit code on our server through the browser, and then host the resulting applications for them? [9] You could run all sorts of services on the servers that these applications could use just by making an API call: making and receiving phone calls, manipulating images, taking credit card payments, etc.

I got so excited about this idea that I couldn't think about anything else. It seemed obvious that this was the future. I didn't particularly want to start another company, but it was clear that this idea would have to be embodied as one, so I decided to move to Cambridge and start it. I hoped to lure Robert into working on it with me, but there I ran into a hitch. Robert was now a postdoc at MIT, and though he'd made a lot of money the last time I'd lured him into working on one of my schemes, it had also

been a huge time sink. So while he agreed that it sounded like a plausible idea, he firmly refused to work on it.

Hmph. Well, I'd do it myself then. I recruited Dan Giffin, who had worked for Viaweb, and two undergrads who wanted summer jobs, and we got to work trying to build what it's now clear is about twenty companies and several open source projects worth of software. The language for defining applications would of course be a dialect of Lisp. But I wasn't so naive as to assume I could spring an overt Lisp on a general audience; we'd hide the parentheses, like Dylan did.

By then there was a name for the kind of company Viaweb was, an "application service provider," or ASP. This name didn't last long before it was replaced by "software as a service," but it was current for long enough that I named this new company after it: it was going to be called Aspra.

I started working on the application builder, Dan worked on network infrastructure, and the two undergrads worked on the first two services (images and phone calls). But about halfway through the summer I realized I really didn't want to run a company — especially not a big one, which it was looking like this would have to be. I'd only started Viaweb because I needed the money. Now that I didn't need money anymore, why was I doing this? If this vision had to be realized as a company, then screw the vision. I'd build a subset that could be done as an open source project.

Much to my surprise, the time I spent working on this stuff was not wasted after all. After we started Y Combinator, I would often encounter startups working on parts of this new architecture, and it was very useful to have spent so much time thinking about it and even trying to write some of it.

The subset I would build as an open source project was the new Lisp, whose parentheses I now wouldn't even have to hide. A lot of Lisp hackers dream of building a new Lisp, partly because one of the distinctive features of the language is that it has dialects, and partly, I think, because we have in our minds a Platonic form of Lisp that all existing dialects fall short of. I certainly did. So at the end of the summer Dan and I switched to working on this new dialect of Lisp, which I called Arc, in a house I bought in Cambridge.

The following spring, lightning struck. I was invited to give a talk at a Lisp conference, so I gave one about how we'd used Lisp at Viaweb. Afterward I put a postscript file of this talk online, on paulgraham.com, which I'd created years before using Viaweb but had never used for anything. In one day it got 30,000 page views. What on earth had happened? The referring urls showed that someone had posted it on Slashdot. [10]

Wow, I thought, there's an audience. If I write something and put it on the web, anyone can read it. That may seem obvious now, but it was surprising then. In the print era there was a narrow channel to readers, guarded by fierce monsters known as editors. The only way to get an audience for anything you wrote was to get it published as a book, or in a newspaper or magazine. Now anyone could publish anything.

This had been possible in principle since 1993, but not many people had realized it yet. I had been intimately involved with building the infrastructure of the web for most of that time, and a writer as well, and it had taken me 8 years to realize it. Even then it took me several years to understand the implications. It meant there would be a whole new generation of essays. [11]

In the print era, the channel for publishing essays had been vanishingly small. Except for a few officially anointed thinkers who went to the right parties in New York, the only people allowed to publish essays were specialists writing about their specialties. There were so many essays that had never been written, because there had been no way to publish them. Now they could be, and I was going to write them. [12]

I've worked on several different things, but to the extent there was a turning point where I figured out what to work on, it was when I started publishing essays online. From then on I knew that whatever else I did, I'd always write essays too.

I knew that online essays would be a marginal medium at first. Socially they'd seem more like rants posted by nutjobs on their GeoCities sites than the genteel and beautifully typeset compositions published in The New Yorker. But by this point I knew enough to find that encouraging instead of discouraging.

One of the most conspicuous patterns I've noticed in my life is how well it has worked, for me at least, to work on things that weren't prestigious. Still life has always been the least prestigious form of painting. Viaweb and Y Combinator both seemed lame when we started them. I still get the glassy eye from strangers when they ask what I'm writing, and I explain that it's an essay I'm going to publish on my web site. Even Lisp, though prestigious intellectually in something like the way Latin is, also seems about as hip.

It's not that unprestigious types of work are good per se. But when you find yourself drawn to some kind of work despite its current lack of prestige, it's a sign both that there's something real to be discovered there, and that you have the right kind of motives. Impure motives are a big danger for the ambitious. If anything is going to lead you astray, it will be the desire to impress people. So while working on things that aren't prestigious doesn't guarantee you're on the right track, it at least guarantees you're not on the most common type of wrong one.

Over the next several years I wrote lots of essays about all kinds of different topics. O'Reilly reprinted a collection of them as a book, called Hackers & Painters after one of the essays in it. I also worked on spam filters, and did some more painting. I used to have dinners for a group of friends every thursday night, which taught me how to cook for groups. And I bought another building in Cambridge, a former candy factory (and later, twas said, porn studio), to use as an office.

One night in October 2003 there was a big party at my house. It was a clever idea of my friend Maria Daniels, who was one of the thursday diners. Three separate hosts would all invite their friends to one party. So for every guest, two thirds of the other guests would be people they didn't know but would probably like. One of the guests was someone I didn't know but would turn out to like a lot: a woman called Jessica Livingston. A couple days later I asked her out.

Jessica was in charge of marketing at a Boston investment bank. This bank thought it understood startups, but over the next year, as she met friends of mine from the startup world, she was surprised how different reality was. And how colorful their stories were. So she decided to compile a book of interviews with startup founders.

When the bank had financial problems and she had to fire half her staff, she started looking for a new job. In early 2005 she interviewed for a marketing job at a Boston VC firm. It took them weeks to make up their minds, and during this time I started telling her about all the things that needed to be fixed about venture capital. They should make a larger number of smaller investments instead of a handful of giant ones, they should be funding younger, more technical founders instead of MBAs, they should let the founders remain as CEO, and so on.

One of my tricks for writing essays had always been to give talks. The prospect of having to stand up in front of a group of people and tell them something that won't waste their time is a great spur to the imagination. When the Harvard Computer Society, the undergrad computer club, asked me to give a talk, I decided I would tell them how to start a startup. Maybe they'd be able to avoid the worst of the mistakes we'd made.

So I gave this talk, in the course of which I told them that the best sources of seed funding were successful startup founders, because then they'd be sources of advice too. Whereupon it seemed they were all looking expectantly at me. Horrified at the prospect of having my inbox flooded by business plans (if I'd only known), I blurted out "But not me!" and went on with the talk. But afterward it occurred to me that I should really stop procrastinating about angel investing. I'd been meaning to since Yahoo bought us, and now it was 7 years later and I still hadn't done one angel investment.

Meanwhile I had been scheming with Robert and Trevor about projects we could work on together. I missed working with them, and it seemed like there had to be something we could collaborate on.

As Jessica and I were walking home from dinner on March 11, at the corner of Garden and Walker streets, these three threads converged. Screw the VCs who were taking so long to make up their minds. We'd start our own investment firm and actually implement the ideas we'd been talking about. I'd fund it, and Jessica could quit her job and work for it, and we'd get Robert and Trevor as partners too. [13]

Once again, ignorance worked in our favor. We had no idea how to be angel investors, and in Boston in 2005 there were no Ron Conways to learn from. So we

just made what seemed like the obvious choices, and some of the things we did turned out to be novel.

There are multiple components to Y Combinator, and we didn't figure them all out at once. The part we got first was to be an angel firm. In those days, those two words didn't go together. There were VC firms, which were organized companies with people whose job it was to make investments, but they only did big, million dollar investments. And there were angels, who did smaller investments, but these were individuals who were usually focused on other things and made investments on the side. And neither of them helped founders enough in the beginning. We knew how helpless founders were in some respects, because we remembered how helpless we'd been. For example, one thing Julian had done for us that seemed to us like magic was to get us set up as a company. We were fine writing fairly difficult software, but actually getting incorporated, with bylaws and stock and all that stuff, how on earth did you do that? Our plan was not only to make seed investments, but to do for startups everything Julian had done for us.

YC was not organized as a fund. It was cheap enough to run that we funded it with our own money. That went right by 99% of readers, but professional investors are thinking "Wow, that means they got all the returns." But once again, this was not due to any particular insight on our part. We didn't know how VC firms were organized. It never occurred to us to try to raise a fund, and if it had, we wouldn't have known where to start. [14]

The most distinctive thing about YC is the batch model: to fund a bunch of startups all at once, twice a year, and then to spend three months focusing intensively on trying to help them. That part we discovered by accident, not merely implicitly but explicitly due to our ignorance about investing. We needed to get experience as investors. What better way, we thought, than to fund a whole bunch of startups at once? We knew undergrads got temporary jobs at tech companies during the summer. Why not organize a summer program where they'd start startups instead? We wouldn't feel guilty for being in a sense fake investors, because they would in a similar sense be fake founders. So while we probably wouldn't make much money out of it, we'd at least get to practice being investors on them, and they for their part would probably have a more interesting summer than they would working at Microsoft.

We'd use the building I owned in Cambridge as our headquarters. We'd all have dinner there once a week — on tuesdays, since I was already cooking for the thursday diners on thursdays — and after dinner we'd bring in experts on startups to give talks.

We knew undergrads were deciding then about summer jobs, so in a matter of days we cooked up something we called the Summer Founders Program, and I posted an announcement on my site, inviting undergrads to apply. I had never imagined that writing essays would be a way to get "deal flow," as investors call it, but it turned out to be the perfect source. [15] We got 225 applications for the Summer Founders Program, and we were surprised to find that a lot of them were from people who'd already graduated, or were about to that spring. Already this SFP thing was starting to feel more serious than we'd intended.

We invited about 20 of the 225 groups to interview in person, and from those we picked 8 to fund. They were an impressive group. That first batch included reddit, Justin Kan and Emmett Shear, who went on to found Twitch, Aaron Swartz, who had already helped write the RSS spec and would a few years later become a martyr for open access, and Sam Altman, who would later become the second president of YC. I don't think it was entirely luck that the first batch was so good. You had to be pretty bold to sign up for a weird thing like the Summer Founders Program instead of a summer job at a legit place like Microsoft or Goldman Sachs.

The deal for startups was based on a combination of the deal we did with Julian ($10k for 10%) and what Robert said MIT grad students got for the summer ($6k). We invested $6k per founder, which in the typical two-founder case was $12k, in return for 6%. That had to be fair, because it was twice as good as the deal we ourselves had taken. Plus that first summer, which was really hot, Jessica brought the founders free air conditioners. [16]

Fairly quickly I realized that we had stumbled upon the way to scale startup funding. Funding startups in batches was more convenient for us, because it meant we could do things for a lot of startups at once, but being part of a batch was better for the startups too. It solved one of the biggest problems faced by founders: the isolation. Now you not only had colleagues, but colleagues who understood the problems you were facing and could tell you how they were solving them.

As YC grew, we started to notice other advantages of scale. The alumni became a tight community, dedicated to helping one another, and especially the current batch, whose shoes they remembered being in. We also noticed that the startups were becoming one another's customers. We used to refer jokingly to the "YC GDP," but as YC grows this becomes less and less of a joke. Now lots of startups get their initial set of customers almost entirely from among their batchmates.

I had not originally intended YC to be a full-time job. I was going to do three things: hack, write essays, and work on YC. As YC grew, and I grew more excited about it, it started to take up a lot more than a third of my attention. But for the first few years I was still able to work on other things.

In the summer of 2006, Robert and I started working on a new version of Arc. This one was reasonably fast, because it was compiled into Scheme. To test this new Arc, I wrote Hacker News in it. It was originally meant to be a news aggregator for startup founders and was called Startup News, but after a few months I got tired of reading about nothing but startups. Plus it wasn't startup founders we wanted to reach. It was future startup founders. So I changed the name to Hacker News and the topic to whatever engaged one's intellectual curiosity.

HN was no doubt good for YC, but it was also by far the biggest source of stress for me. If all I'd had to do was select and help founders, life would have been so easy. And that implies that HN was a mistake. Surely the biggest source of stress in one's work should at least be something close to the core of the work. Whereas I was like someone who was in pain while running a marathon not from the exertion of running, but because I had a blister from an ill-fitting shoe. When I was dealing with some urgent problem during YC, there was about a 60% chance it had to do with HN, and a 40% chance it had do with everything else combined. [17]

As well as HN, I wrote all of YC's internal software in Arc. But while I continued to work a good deal in Arc, I gradually stopped working on Arc, partly because I didn't have time to, and partly because it was a lot less attractive to mess around with the language now that we had all this infrastructure depending on it. So now my three projects were reduced to two: writing essays and working on YC.

YC was different from other kinds of work I've done. Instead of deciding for myself what to work on, the problems came to me. Every 6 months there was a new batch of startups, and their problems, whatever they were, became our problems. It was very engaging work, because their problems were quite varied, and the good founders were very effective. If you were trying to learn the most you could about startups in the shortest possible time, you couldn't have picked a better way to do it.

There were parts of the job I didn't like. Disputes between cofounders, figuring out when people were lying to us, fighting with people who maltreated the startups, and so on. But I worked hard even at the parts I didn't like. I was haunted by something Kevin Hale once said about companies: "No one works harder than the boss." He meant it both descriptively and prescriptively, and it was the second part that scared me. I wanted YC to be good, so if how hard I worked set the upper bound on how hard everyone else worked, I'd better work very hard.

One day in 2010, when he was visiting California for interviews, Robert Morris did something astonishing: he offered me unsolicited advice. I can only remember him doing that once before. One day at Viaweb, when I was bent over double from a kidney stone, he suggested that it would be a good idea for him to take me to the hospital. That was what it took for Rtm to offer unsolicited advice. So I remember his exact words very clearly. "You know," he said, "you should make sure Y Combinator isn't the last cool thing you do."

At the time I didn't understand what he meant, but gradually it dawned on me that he was saying I should quit. This seemed strange advice, because YC was doing great. But if there was one thing rarer than Rtm offering advice, it was Rtm being wrong. So this set me thinking. It was true that on my current trajectory, YC would be the last thing I did, because it was only taking up more of my attention. It had already eaten Arc, and was in the process of eating essays too. Either YC was my life's work or I'd have to leave eventually. And it wasn't, so I would.

In the summer of 2012 my mother had a stroke, and the cause turned out to be a blood clot caused by colon cancer. The stroke destroyed her balance, and she was put in a nursing home, but she really wanted to get out of it and back to her house, and my sister and I were determined to help her do it. I used to fly up to Oregon to visit her regularly, and I had a lot of time to think on those flights. On one of them I realized I was ready to hand YC over to someone else.

I asked Jessica if she wanted to be president, but she didn't, so we decided we'd try to recruit Sam Altman. We talked to Robert and Trevor and we agreed to make it a complete changing of the guard. Up till that point YC had been controlled by the original LLC we four had started. But we wanted YC to last for a long time, and to do that it couldn't be controlled by the founders. So if Sam said yes, we'd let him reorganize YC. Robert and I would retire, and Jessica and Trevor would become ordinary partners.

When we asked Sam if he wanted to be president of YC, initially he said no. He wanted to start a startup to make nuclear reactors. But I kept at it, and in October 2013 he finally agreed. We decided he'd take over starting with the winter 2014 batch. For the rest of 2013 I left running YC more and more to Sam, partly so he could learn the job, and partly because I was focused on my mother, whose cancer had returned.

She died on January 15, 2014. We knew this was coming, but it was still hard when it did.

I kept working on YC till March, to help get that batch of startups through Demo Day, then I checked out pretty completely. (I still talk to alumni and to new startups working on things I'm interested in, but that only takes a few hours a week.)

What should I do next? Rtm's advice hadn't included anything about that. I wanted to do something completely different, so I decided I'd paint. I wanted to see how good I could get if I really focused on it. So the day after I stopped working on YC, I started painting. I was rusty and it took a while to get back into shape, but it was at least completely engaging. [18]

I spent most of the rest of 2014 painting. I'd never been able to work so uninterruptedly before, and I got to be better than I had been. Not good enough, but better. Then in November, right in the middle of a painting, I ran out of steam. Up till that point I'd always been curious to see how the painting I was working on would turn out, but suddenly finishing this one seemed like a chore. So I stopped working on it and cleaned my brushes and haven't painted since. So far anyway.

I realize that sounds rather wimpy. But attention is a zero sum game. If you can choose what to work on, and you choose a project that's not the best one (or at least a good one) for you, then it's getting in the way of another project that is. And at 50 there was some opportunity cost to screwing around.

I started writing essays again, and wrote a bunch of new ones over the next few months. I even wrote a couple that weren't about startups. Then in March 2015 I started working on Lisp again.

The distinctive thing about Lisp is that its core is a language defined by writing an interpreter in itself. It wasn't originally intended as a programming language in the ordinary sense. It was meant to be a formal model of computation, an alternative to the Turing machine. If you want to write an interpreter for a language in itself, what's the minimum set of predefined operators you need? The Lisp that John McCarthy invented, or more accurately discovered, is an answer to that question. [19]

McCarthy didn't realize this Lisp could even be used to program computers till his grad student Steve Russell suggested it. Russell translated McCarthy's interpreter into IBM 704 machine language, and from that point Lisp started also to be a programming language in the ordinary sense. But its origins as a model of computation gave it a power and elegance that other languages couldn't match. It was this that attracted me in college, though I didn't understand why at the time.

McCarthy's 1960 Lisp did nothing more than interpret Lisp expressions. It was missing a lot of things you'd want in a programming language. So these had to be added, and when they were, they weren't defined using McCarthy's original axiomatic approach. That wouldn't have been feasible at the time. McCarthy tested his interpreter by hand-simulating the execution of programs. But it was already getting close to the limit of interpreters you could test that way — indeed, there was a bug in it that McCarthy had overlooked. To test a more complicated interpreter, you'd have had to run it, and computers then weren't powerful enough.

Now they are, though. Now you could continue using McCarthy's axiomatic approach till you'd defined a complete programming language. And as long as every

change you made to McCarthy's Lisp was a discoveredness-preserving transformation, you could, in principle, end up with a complete language that had this quality. Harder to do than to talk about, of course, but if it was possible in principle, why not try? So I decided to take a shot at it. It took 4 years, from March 26, 2015 to October 12, 2019. It was fortunate that I had a precisely defined goal, or it would have been hard to keep at it for so long.

I wrote this new Lisp, called Bel, in itself in Arc. That may sound like a contradiction, but it's an indication of the sort of trickery I had to engage in to make this work. By means of an egregious collection of hacks I managed to make something close enough to an interpreter written in itself that could actually run. Not fast, but fast enough to test.

I had to ban myself from writing essays during most of this time, or I'd never have finished. In late 2015 I spent 3 months writing essays, and when I went back to working on Bel I could barely understand the code. Not so much because it was badly written as because the problem is so convoluted. When you're working on an interpreter written in itself, it's hard to keep track of what's happening at what level, and errors can be practically encrypted by the time you get them.

So I said no more essays till Bel was done. But I told few people about Bel while I was working on it. So for years it must have seemed that I was doing nothing, when in fact I was working harder than I'd ever worked on anything. Occasionally after wrestling for hours with some gruesome bug I'd check Twitter or HN and see someone asking "Does Paul Graham still code?"

Working on Bel was hard but satisfying. I worked on it so intensively that at any given time I had a decent chunk of the code in my head and could write more there. I remember taking the boys to the coast on a sunny day in 2015 and figuring out how to deal with some problem involving continuations while I watched them play in the tide pools. It felt like I was doing life right. I remember that because I was slightly dismayed at how novel it felt. The good news is that I had more moments like this over the next few years.

In the summer of 2016 we moved to England. We wanted our kids to see what it was like living in another country, and since I was a British citizen by birth, that seemed

the obvious choice. We only meant to stay for a year, but we liked it so much that we still live there. So most of Bel was written in England.

In the fall of 2019, Bel was finally finished. Like McCarthy's original Lisp, it's a spec rather than an implementation, although like McCarthy's Lisp it's a spec expressed as code.

Now that I could write essays again, I wrote a bunch about topics I'd had stacked up. I kept writing essays through 2020, but I also started to think about other things I could work on. How should I choose what to do? Well, how had I chosen what to work on in the past? I wrote an essay for myself to answer that question, and I was surprised how long and messy the answer turned out to be. If this surprised me, who'd lived it, then I thought perhaps it would be interesting to other people, and encouraging to those with similarly messy lives. So I wrote a more detailed version for others to read, and this is the last sentence of it.

Notes

[1] My experience skipped a step in the evolution of computers: time-sharing machines with interactive OSes. I went straight from batch processing to microcomputers, which made microcomputers seem all the more exciting.

[2] Italian words for abstract concepts can nearly always be predicted from their English cognates (except for occasional traps like polluzione). It's the everyday words that differ. So if you string together a lot of abstract concepts with a few simple verbs, you can make a little Italian go a long way.

[3] I lived at Piazza San Felice 4, so my walk to the Accademia went straight down the spine of old Florence: past the Pitti, across the bridge, past Orsanmichele, between the Duomo and the Baptistery, and then up Via Ricasoli to Piazza San Marco. I saw Florence at street level in every possible condition, from empty dark winter evenings to sweltering summer days when the streets were packed with tourists.

[4] You can of course paint people like still lives if you want to, and they're willing. That sort of portrait is arguably the apex of still life painting, though the long sitting does tend to produce pained expressions in the sitters.

[5] Interleaf was one of many companies that had smart people and built impressive technology, and yet got crushed by Moore's Law. In the 1990s the exponential growth in the power of commodity (i.e. Intel) processors rolled up high-end, special-purpose hardware and software companies like a bulldozer.

[6] The signature style seekers at RISD weren't specifically mercenary. In the art world, money and coolness are tightly coupled. Anything expensive comes to be seen as cool, and anything seen as cool will soon become equally expensive.

[7] Technically the apartment wasn't rent-controlled but rent-stabilized, but this is a refinement only New Yorkers would know or care about. The point is that it was really cheap, less than half market price.

[8] Most software you can launch as soon as it's done. But when the software is an online store builder and you're hosting the stores, if you don't have any users yet, that fact will be painfully obvious. So before we could launch publicly we had to launch privately, in the sense of recruiting an initial set of users and making sure they had decent-looking stores.

[9] We'd had a code editor in Viaweb for users to define their own page styles. They didn't know it, but they were editing Lisp expressions underneath. But this wasn't an app editor, because the code ran when the merchants' sites were generated, not when shoppers visited them.

[10] This was the first instance of what is now a familiar experience, and so was what happened next, when I read the comments and found they were full of angry people. How could I claim that Lisp was better than other languages? Weren't they all Turing complete? People who see the responses to essays I write sometimes tell me how sorry they feel for me, but I'm not exaggerating when I reply that it has always been like this, since the very beginning. It comes with the territory. An essay must tell readers things they don't already know, and some people dislike being told such things.

[11] People put plenty of stuff on the internet in the 90s of course, but putting something online is not the same as publishing it online. Publishing online means you treat the online version as the (or at least a) primary version.

[12] There is a general lesson here that our experience with Y Combinator also teaches: Customs continue to constrain you long after the restrictions that caused them have disappeared. Customary VC practice had once, like the customs about publishing essays, been based on real constraints. Startups had once been much more expensive to start, and proportionally rare. Now they could be cheap and common, but the VCs' customs still reflected the old world, just as customs about writing essays still reflected the constraints of the print era.

Which in turn implies that people who are independent-minded (i.e. less influenced by custom) will have an advantage in fields affected by rapid change (where customs are more likely to be obsolete).

Here's an interesting point, though: you can't always predict which fields will be affected by rapid change. Obviously software and venture capital will be, but who would have predicted that essay writing would be?

[13] Y Combinator was not the original name. At first we were called Cambridge Seed. But we didn't want a regional name, in case someone copied us in Silicon Valley, so we renamed ourselves after one of the coolest tricks in the lambda calculus, the Y combinator.

I picked orange as our color partly because it's the warmest, and partly because no VC used it. In 2005 all the VCs used staid colors like maroon, navy blue, and forest green, because they were trying to appeal to LPs, not founders. The YC logo itself is an inside joke: the Viaweb logo had been a white V on a red circle, so I made the YC logo a white Y on an orange square.

[14] YC did become a fund for a couple years starting in 2009, because it was getting so big I could no longer afford to fund it personally. But after Heroku got bought we had enough money to go back to being self-funded.

[15] I've never liked the term "deal flow," because it implies that the number of new startups at any given time is fixed. This is not only false, but it's the purpose of YC to falsify it, by causing startups to be founded that would not otherwise have existed.

[16] She reports that they were all different shapes and sizes, because there was a run on air conditioners and she had to get whatever she could, but that they were all heavier than she could carry now.

[17] Another problem with HN was a bizarre edge case that occurs when you both write essays and run a forum. When you run a forum, you're assumed to see if not every conversation, at least every conversation involving you. And when you write essays, people post highly imaginative misinterpretations of them on forums. Individually these two phenomena are tedious but bearable, but the combination is disastrous. You actually have to respond to the misinterpretations, because the assumption that you're present in the conversation means that not responding to any sufficiently upvoted misinterpretation reads as a tacit admission that it's correct. But that in turn encourages more; anyone who wants to pick a fight with you senses that now is their chance.

[18] The worst thing about leaving YC was not working with Jessica anymore. We'd been working on YC almost the whole time we'd known each other, and we'd neither tried nor wanted to separate it from our personal lives, so leaving was like pulling up a deeply rooted tree.

[19] One way to get more precise about the concept of invented vs discovered is to talk about space aliens. Any sufficiently advanced alien civilization would certainly know about the Pythagorean theorem, for example. I believe, though with less certainty, that they would also know about the Lisp in McCarthy's 1960 paper.

But if so there's no reason to suppose that this is the limit of the language that might be known to them. Presumably aliens need numbers and errors and I/O too. So it seems likely there exists at least one path out of McCarthy's Lisp along which discoveredness is preserved.

Thanks to Trevor Blackwell, John Collison, Patrick Collison, Daniel Gackle, Ralph Hazell, Jessica Livingston, Robert Morris, and Harj Taggar for reading drafts of this.