

# Great Hackers

paulgraham.com · Paul Graham · 2004-07 · [source](#)

---

|

July 2004

(This essay is derived from a talk at Oscon 2004.)

A few months ago I finished a new book, and in reviews I keep noticing words like "provocative" and "controversial." To say nothing of "idiotic."

I didn't mean to make the book controversial. I was trying to make it efficient. I didn't want to waste people's time telling them things they already knew. It's more efficient just to give them the diffs. But I suppose that's bound to yield an alarming book.

Edisons

There's no controversy about which idea is most controversial: the suggestion that variation in wealth might not be as big a problem as we think.

I didn't say in the book that variation in wealth was in itself a good thing. I said in some situations it might be a sign of good things. A throbbing headache is not a good thing, but it can be a sign of a good thing-- for example, that you're recovering consciousness after being hit on the head.

Variation in wealth can be a sign of variation in productivity. (In a society of one, they're identical.) And that is almost certainly a good thing: if your society has no variation in productivity, it's probably not because everyone is Thomas Edison. It's probably because you have no Thomas Edisons.

In a low-tech society you don't see much variation in productivity. If you have a tribe of nomads collecting sticks for a fire, how much more productive is the best stick gatherer going to be than the worst? A factor of two? Whereas when you hand people a complex tool like a computer, the variation in what they can do with it is enormous.

That's not a new idea. Fred Brooks wrote about it in 1974, and the study he quoted was published in 1968. But I think he underestimated the variation between programmers. He wrote about productivity in lines of code: the best programmers can solve a given problem in a tenth the time. But what if the problem isn't given? In programming, as in many fields, the hard part isn't solving problems, but deciding what problems to solve. Imagination is hard to measure, but in practice it dominates the kind of productivity that's measured in lines of code.

Productivity varies in any field, but there are few in which it varies so much. The variation between programmers is so great that it becomes a difference in kind. I don't think this is something intrinsic to programming, though. In every field, technology magnifies differences in productivity. I think what's happening in programming is just that we have a lot of technological leverage. But in every field the lever is getting longer, so the variation we see is something that more and more fields will see as time goes on. And the success of companies, and countries, will depend increasingly on how they deal with it.

If variation in productivity increases with technology, then the contribution of the most productive individuals will not only be disproportionately large, but will actually grow with time. When you reach the point where 90% of a group's output is created by 1% of its members, you lose big if something (whether Viking raids, or central planning) drags their productivity down to the average.

If we want to get the most out of them, we need to understand these especially productive people. What motivates them? What do they need to do their jobs? How do you recognize them? How do you get them to come and work for you? And then of course there's the

question, how do you become one?

## More than Money

I know a handful of super-hackers, so I sat down and thought about what they have in common. Their defining quality is probably that they really love to program. Ordinary programmers write code to pay the bills. Great hackers think of it as something they do for fun, and which they're delighted to find people will pay them for.

Great programmers are sometimes said to be indifferent to money. This isn't quite true. It is true that all they really care about is doing interesting work. But if you make enough money, you get to work on whatever you want, and for that reason hackers are attracted by the idea of making really large amounts of money. But as long as they still have to show up for work every day, they care more about what they do there than how much they get paid for it.

Economically, this is a fact of the greatest importance, because it means you don't have to pay great hackers anything like what they're worth. A great programmer might be ten or a hundred times as productive as an ordinary one, but he'll consider himself lucky to get paid three times as much. As I'll explain later, this is partly because great hackers don't know how good they are. But it's also because money is not the main thing they want.

What do hackers want? Like all craftsmen, hackers like good tools. In fact, that's an understatement. Good hackers find it unbearable to use bad tools. They'll simply refuse to work on projects with the wrong infrastructure.

At a startup I once worked for, one of the things pinned up on our bulletin board was an ad from IBM. It was a picture of an AS400, and the headline read, I think, "hackers despise it." [1]

When you decide what infrastructure to use for a project, you're not just making a technical decision. You're also making a social decision, and this may be the more important of the two. For

example, if your company wants to write some software, it might seem a prudent choice to write it in Java. But when you choose a language, you're also choosing a community. The programmers you'll be able to hire to work on a Java project won't be as smart as the ones you could get to work on a project written in Python. And the quality of your hackers probably matters more than the language you choose. Though, frankly, the fact that good hackers prefer Python to Java should tell you something about the relative merits of those languages.

Business types prefer the most popular languages because they view languages as standards. They don't want to bet the company on Betamax. The thing about languages, though, is that they're not just standards. If you have to move bits over a network, by all means use TCP/IP. But a programming language isn't just a format. A programming language is a medium of expression.

I've read that Java has just overtaken Cobol as the most popular language. As a standard, you couldn't wish for more. But as a medium of expression, you could do a lot better. Of all the great programmers I can think of, I know of only one who would voluntarily program in Java. And of all the great programmers I can think of who don't work for Sun, on Java, I know of zero.

Great hackers also generally insist on using open source software. Not just because it's better, but because it gives them more control. Good hackers insist on control. This is part of what makes them good hackers: when something's broken, they need to fix it. You want them to feel this way about the software they're writing for you. You shouldn't be surprised when they feel the same way about the operating system.

A couple years ago a venture capitalist friend told me about a new startup he was involved with. It sounded promising. But the next time I talked to him, he said they'd decided to build their software on Windows NT, and had just hired a very experienced NT developer to be their chief technical officer. When I heard this, I thought, these guys are doomed. One, the CTO couldn't be a first rate hacker, because to become an eminent NT developer he would have

had to use NT voluntarily, multiple times, and I couldn't imagine a great hacker doing that; and two, even if he was good, he'd have a hard time hiring anyone good to work for him if the project had to be built on NT. [2]

## The Final Frontier

After software, the most important tool to a hacker is probably his office. Big companies think the function of office space is to express rank. But hackers use their offices for more than that: they use their office as a place to think in. And if you're a technology company, their thoughts are your product. So making hackers work in a noisy, distracting environment is like having a paint factory where the air is full of soot.

The cartoon strip Dilbert has a lot to say about cubicles, and with good reason. All the hackers I know despise them. The mere prospect of being interrupted is enough to prevent hackers from working on hard problems. If you want to get real work done in an office with cubicles, you have two options: work at home, or come in early or late or on a weekend, when no one else is there. Don't companies realize this is a sign that something is broken? An office environment is supposed to be something that helps you work, not something you work despite.

Companies like Cisco are proud that everyone there has a cubicle, even the CEO. But they're not so advanced as they think; obviously they still view office space as a badge of rank. Note too that Cisco is famous for doing very little product development in house. They get new technology by buying the startups that created it-- where presumably the hackers did have somewhere quiet to work.

One big company that understands what hackers need is Microsoft. I once saw a recruiting ad for Microsoft with a big picture of a door. Work for us, the premise was, and we'll give you a place to work where you can actually get work done. And you know, Microsoft is remarkable among big companies in that they are able to develop software in house. Not well, perhaps, but well enough.

If companies want hackers to be productive, they should look at

what they do at home. At home, hackers can arrange things themselves so they can get the most done. And when they work at home, hackers don't work in noisy, open spaces; they work in rooms with doors. They work in cosy, neighborhoody places with people around and somewhere to walk when they need to mull something over, instead of in glass boxes set in acres of parking lots. They have a sofa they can take a nap on when they feel tired, instead of sitting in a coma at their desk, pretending to work. There's no crew of people with vacuum cleaners that roars through every evening during the prime hacking hours. There are no meetings or, God forbid, corporate retreats or team-building exercises. And when you look at what they're doing on that computer, you'll find it reinforces what I said earlier about tools. They may have to use Java and Windows at work, but at home, where they can choose for themselves, you're more likely to find them using Perl and Linux.

Indeed, these statistics about Cobol or Java being the most popular language can be misleading. What we ought to look at, if we want to know what tools are best, is what hackers choose when they can choose freely-- that is, in projects of their own. When you ask that question, you find that open source operating systems already have a dominant market share, and the number one language is probably Perl.

### Interesting

Along with good tools, hackers want interesting projects. What makes a project interesting? Well, obviously overtly sexy applications like stealth planes or special effects software would be interesting to work on. But any application can be interesting if it poses novel technical challenges. So it's hard to predict which problems hackers will like, because some become interesting only when the people working on them discover a new kind of solution. Before ITA (who wrote the software inside Orbitz), the people working on airline fare searches probably thought it was one of the most boring applications imaginable. But ITA made it interesting by redefining the problem in a more ambitious way.

I think the same thing happened at Google. When Google was founded, the conventional wisdom among the so-called portals was that search was boring and unimportant. But the guys at Google didn't think search was boring, and that's why they do it so well.

This is an area where managers can make a difference. Like a parent saying to a child, I bet you can't clean up your whole room in ten minutes, a good manager can sometimes redefine a problem as a more interesting one. Steve Jobs seems to be particularly good at this, in part simply by having high standards. There were a lot of small, inexpensive computers before the Mac. He redefined the problem as: make one that's beautiful. And that probably drove the developers harder than any carrot or stick could.

They certainly delivered. When the Mac first appeared, you didn't even have to turn it on to know it would be good; you could tell from the case. A few weeks ago I was walking along the street in Cambridge, and in someone's trash I saw what appeared to be a Mac carrying case. I looked inside, and there was a Mac SE. I carried it home and plugged it in, and it booted. The happy Macintosh face, and then the finder. My God, it was so simple. It was just like ... Google.

Hackers like to work for people with high standards. But it's not enough just to be exacting. You have to insist on the right things. Which usually means that you have to be a hacker yourself. I've seen occasional articles about how to manage programmers. Really there should be two articles: one about what to do if you are yourself a programmer, and one about what to do if you're not. And the second could probably be condensed into two words: give up.

The problem is not so much the day to day management. Really good hackers are practically self-managing. The problem is, if you're not a hacker, you can't tell who the good hackers are. A similar problem explains why American cars are so ugly. I call it the design paradox. You might think that you could make your products beautiful just by hiring a great designer to design them. But if you yourself don't have good taste, how are you going to recognize a good designer? By definition you can't tell from his portfolio.

And you can't go by the awards he's won or the jobs he's had, because in design, as in most fields, those tend to be driven by fashion and schmoozing, with actual ability a distant third. There's no way around it: you can't manage a process intended to produce beautiful things without knowing what beautiful is. American cars are ugly because American car companies are run by people with bad taste.

Many people in this country think of taste as something elusive, or even frivolous. It is neither. To drive design, a manager must be the most demanding user of a company's products. And if you have really good taste, you can, as Steve Jobs does, make satisfying you the kind of problem that good people like to work on.

### Nasty Little Problems

It's pretty easy to say what kinds of problems are not interesting: those where instead of solving a few big, clear, problems, you have to solve a lot of nasty little ones. One of the worst kinds of projects is writing an interface to a piece of software that's full of bugs. Another is when you have to customize something for an individual client's complex and ill-defined needs. To hackers these kinds of projects are the death of a thousand cuts.

The distinguishing feature of nasty little problems is that you don't learn anything from them. Writing a compiler is interesting because it teaches you what a compiler is. But writing an interface to a buggy piece of software doesn't teach you anything, because the bugs are random. [3] So it's not just fastidiousness that makes good hackers avoid nasty little problems. It's more a question of self-preservation. Working on nasty little problems makes you stupid. Good hackers avoid it for the same reason models avoid cheeseburgers.

Of course some problems inherently have this character. And because of supply and demand, they pay especially well. So a company that found a way to get great hackers to work on tedious problems would be very successful. How would you do it?

One place this happens is in startups. At our startup we had Robert Morris working as a system administrator. That's like having the Rolling Stones play at a bar mitzvah. You can't hire that kind of talent. But people will do any amount of drudgery for companies of which they're the founders. [4]

Bigger companies solve the problem by partitioning the company. They get smart people to work for them by establishing a separate R&D department where employees don't have to work directly on customers' nasty little problems. [5] In this model, the research department functions like a mine. They produce new ideas; maybe the rest of the company will be able to use them.

You may not have to go to this extreme.

Bottom-up programming

suggests another way to partition the company: have the smart people work as toolmakers. If your company makes software to do x, have one group that builds tools for writing software of that type, and another that uses these tools to write the applications. This way you might be able to get smart people to write 99% of your code, but still keep them almost as insulated from users as they would be in a traditional research department. The toolmakers would have users, but they'd only be the company's own developers. [6]

If Microsoft used this approach, their software wouldn't be so full of security holes, because the less smart people writing the actual applications wouldn't be doing low-level stuff like allocating memory. Instead of writing Word directly in C, they'd be plugging together big Lego blocks of Word-language. (Duplo, I believe, is the technical term.)

Clumping

Along with interesting problems, what good hackers like is other good hackers. Great hackers tend to clump together-- sometimes spectacularly so, as at Xerox Parc. So you won't attract good hackers in linear proportion to how good an environment you create for them. The tendency to clump means it's more like the square of the environment. So it's winner take all. At any given time, there are only about ten or twenty places where hackers most want to

work, and if you aren't one of them, you won't just have fewer great hackers, you'll have zero.

Having great hackers is not, by itself, enough to make a company successful. It works well for Google and ITA, which are two of the hot spots right now, but it didn't help Thinking Machines or Xerox. Sun had a good run for a while, but their business model is a down elevator. In that situation, even the best hackers can't save you.

I think, though, that all other things being equal, a company that can attract great hackers will have a huge advantage. There are people who would disagree with this. When we were making the rounds of venture capital firms in the 1990s, several told us that software companies didn't win by writing great software, but through brand, and dominating channels, and doing the right deals.

They really seemed to believe this, and I think I know why. I think what a lot of VCs are looking for, at least unconsciously, is the next Microsoft. And of course if Microsoft is your model, you shouldn't be looking for companies that hope to win by writing great software. But VCs are mistaken to look for the next Microsoft, because no startup can be the next Microsoft unless some other company is prepared to bend over at just the right moment and be the next IBM.

It's a mistake to use Microsoft as a model, because their whole culture derives from that one lucky break. Microsoft is a bad data point. If you throw them out, you find that good products do tend to win in the market. What VCs should be looking for is the next Apple, or the next Google.

I think Bill Gates knows this. What worries him about Google is not the power of their brand, but the fact that they have better hackers. [7]

## Recognition

So who are the great hackers? How do you know when you meet one? That turns out to be very hard. Even hackers can't tell. I'm

pretty sure now that my friend Trevor Blackwell is a great hacker. You may have read on Slashdot how he made his own Segway. The remarkable thing about this project was that he wrote all the software in one day (in Python, incidentally).

For Trevor, that's par for the course. But when I first met him, I thought he was a complete idiot. He was standing in Robert Morris's office babbling at him about something or other, and I remember standing behind him making frantic gestures at Robert to shoo this nut out of his office so we could go to lunch. Robert says he misjudged Trevor at first too. Apparently when Robert first met him, Trevor had just begun a new scheme that involved writing down everything about every aspect of his life on a stack of index cards, which he carried with him everywhere. He'd also just arrived from Canada, and had a strong Canadian accent and a mullet.

The problem is compounded by the fact that hackers, despite their reputation for social obliviousness, sometimes put a good deal of effort into seeming smart. When I was in grad school I used to hang around the MIT AI Lab occasionally. It was kind of intimidating at first. Everyone there spoke so fast. But after a while I learned the trick of speaking fast. You don't have to think any faster; just use twice as many words to say everything.

With this amount of noise in the signal, it's hard to tell good hackers when you meet them. I can't tell, even now. You also can't tell from their resumes. It seems like the only way to judge a hacker is to work with him on something.

And this is the reason that high-tech areas only happen around universities. The active ingredient here is not so much the professors as the students. Startups grow up around universities because universities bring together promising young people and make them work on the same projects. The smart ones learn who the other smart ones are, and together they cook up new projects of their own.

Because you can't tell a great hacker except by working with him,

hackers themselves can't tell how good they are. This is true to a degree in most fields. I've found that people who are great at something are not so much convinced of their own greatness as mystified at why everyone else seems so incompetent.

But it's particularly hard for hackers to know how good they are, because it's hard to compare their work. This is easier in most other fields. In the hundred meters, you know in 10 seconds who's fastest. Even in math there seems to be a general consensus about which problems are hard to solve, and what constitutes a good solution. But hacking is like writing. Who can say which of two novels is better? Certainly not the authors.

With hackers, at least, other hackers can tell. That's because, unlike novelists, hackers collaborate on projects. When you get to hit a few difficult problems over the net at someone, you learn pretty quickly how hard they hit them back. But hackers can't watch themselves at work. So if you ask a great hacker how good he is, he's almost certain to reply, I don't know. He's not just being modest. He really doesn't know.

And none of us know, except about people we've actually worked with. Which puts us in a weird situation: we don't know who our heroes should be. The hackers who become famous tend to become famous by random accidents of PR. Occasionally I need to give an example of a great hacker, and I never know who to use. The first names that come to mind always tend to be people I know personally, but it seems lame to use them. So, I think, maybe I should say Richard Stallman, or Linus Torvalds, or Alan Kay, or someone famous like that. But I have no idea if these guys are great hackers. I've never worked with them on anything.

If there is a Michael Jordan of hacking, no one knows, including him.

## Cultivation

Finally, the question the hackers have all been wondering about: how do you become a great hacker? I don't know if it's possible to make yourself into one. But it's certainly possible to do things

that make you stupid, and if you can make yourself stupid, you can probably make yourself smart too.

The key to being a good hacker may be to work on what you like. When I think about the great hackers I know, one thing they have in common is the extreme difficulty of making them work on anything they don't want to. I don't know if this is cause or effect; it may be both.

To do something well you have to love it. So to the extent you can preserve hacking as something you love, you're likely to do it well. Try to keep the sense of wonder you had about programming at age 14. If you're worried that your current job is rotting your brain, it probably is.

The best hackers tend to be smart, of course, but that's true in a lot of fields. Is there some quality that's unique to hackers? I asked some friends, and the number one thing they mentioned was curiosity. I'd always supposed that all smart people were curious-- that curiosity was simply the first derivative of knowledge. But apparently hackers are particularly curious, especially about how things work. That makes sense, because programs are in effect giant descriptions of how things work.

Several friends mentioned hackers' ability to concentrate-- their ability, as one put it, to "tune out everything outside their own heads." I've certainly noticed this. And I've heard several hackers say that after drinking even half a beer they can't program at all. So maybe hacking does require some special ability to focus. Perhaps great hackers can load a large amount of context into their head, so that when they look at a line of code, they see not just that line but the whole program around it. John McPhee wrote that Bill Bradley's success as a basketball player was due partly to his extraordinary peripheral vision. "Perfect" eyesight means about 47 degrees of vertical peripheral vision. Bill Bradley had 70; he could see the basket when he was looking at the floor.

Maybe great hackers have some similar inborn ability. (I cheat by using a very dense language, which shrinks the court.)

This could explain the disconnect over cubicles. Maybe the people in charge of facilities, not having any concentration to shatter, have no idea that working in a cubicle feels to a hacker like having one's brain in a blender. (Whereas Bill, if the rumors of autism are true, knows all too well.)

One difference I've noticed between great hackers and smart people in general is that hackers are more politically incorrect. To the extent there is a secret handshake among good hackers, it's when they know one another well enough to express opinions that would get them stoned to death by the general public. And I can see why political incorrectness would be a useful quality in programming. Programs are very complex and, at least in the hands of good programmers, very fluid. In such situations it's helpful to have a habit of questioning assumptions.

Can you cultivate these qualities? I don't know. But you can at least not repress them. So here is my best shot at a recipe. If it is possible to make yourself into a great hacker, the way to do it may be to make the following deal with yourself: you never have to work on boring projects (unless your family will starve otherwise), and in return, you'll never allow yourself to do a half-assed job. All the great hackers I know seem to have made that deal, though perhaps none of them had any choice in the matter.

## Notes

[1] In fairness, I have to say that IBM makes decent hardware. I wrote this on an IBM laptop.

[2] They did turn out to be doomed. They shut down a few months

later.

[3] I think this is what people mean when they talk about the "meaning of life." On the face of it, this seems an odd idea. Life isn't an expression; how could it have meaning? But it can have a quality that feels a lot like meaning. In a project like a compiler, you have to solve a lot of problems, but the problems all fall into a pattern, as in a signal. Whereas when the problems you have to solve are random, they seem like noise.

[4] Einstein at one point worked designing refrigerators. (He had equity.)

[5] It's hard to say exactly what constitutes research in the computer world, but as a first approximation, it's software that doesn't have users.

I don't think it's publication that makes the best hackers want to work in research departments. I think it's mainly not having to have a three hour meeting with a product manager about problems integrating the Korean version of Word 13.27 with the talking paperclip.

[6] Something similar has been happening for a long time in the construction industry. When you had a house built a couple hundred years ago, the local builders built everything in it. But increasingly what builders do is assemble components designed and manufactured by someone else. This has, like the arrival of desktop publishing, given people the freedom to experiment in disastrous ways, but it is certainly more efficient.

[7] Google is much more dangerous to Microsoft than Netscape was. Probably more dangerous than any other company has ever been. Not least because they're determined to fight. On their job listing page, they say that one of their "core values" is "Don't be evil." From a company selling soybean oil or mining equipment, such a statement would merely be eccentric. But I think all of us in the computer world recognize who that is a declaration of war on.

Thanks to Jessica Livingston, Robert Morris, and Sarah Harlin for reading earlier versions of this talk.

