The Best of edw519

A Hacker News Top Contributor

by Ed Weissman

Contents

Forward

Chapter 1 - Advice to Young Programmers

Chapter 2 - Education

Chapter 3 - Careers

Chapter 4 - Work Habits

68. Can programming be boring?
69. How do you manage your time spent?
70. How do you capture good ideas?
71. How do you get good at programming?
72. How do I rise out of the ordinary?
73. You're Not the Problem, the Work Is
74. What are the basic modules in your library
75. Why are you a "caveman" programmer?
76. What tools do you use for analysis?
77. Every Task is Complete or Not Complete
78. Why do you love email?
79. Where do you like to sit in your office?
80. How important is office space?
81. How do you pick the best language for you?
82. What are the advantages of working at home
83. Any other advantages to working at home?
84. How do manage to enjoy working at home?
85. How do you split your programming time?
86. Working at the Library

Chapter 5 - The Programmer's Lifestyle

87. What got you "hooked"?
88. What Matters Most to a Programmer
89. Is programming hard work?
90. Why I Do Not Feel Like a Fraud
91. How do I become addicted to programming?
92. Why do we lose our passion?
93. Why are languages so unimportant?

Chapter 6 - Philosophy

120. Do you think you have peaked?
121. Why do you program?
122. What are your hardest learned lessons?
123. What have you learned from mentors?
124. Who are the real heros of programming?
125. Relentlessness
126. What if I'm not as good as someone else?
127. It's Never Too Late
128. It Can't Be Done
129. How perfect do you have to be?
130. Are good programmers born or made?
131. What's your greatest life lesson?
132. Are programmers expensive?
133. How far from shore are you?
134. What is "fear of failure"?
135. The Code is the Star
136. How can I be excellent with a day job?
137. How do you put your skills to good?
138. Issues vs. Details
139. How is an issue different from a detail?
140. Living in Two Worlds
141. What are the biggest programming myths?
142. Don't Pull a Teddy Roosevelt
143. Should I learn or build first?
144. What's the big deal with startups?
145. How do you find inspiration?
146. How to Never Say "No"
147. Why I'm a Late Bloomer
148. How does one turn out the way they do?
149. How important to society is your software?

Chapter 7 - Building Stuff

176. Technical Debt
177. Annoyances vs. Requirements
178. What can be optimized?
179. Why pre-develop?
180. Documentation Belongs in the Code
181. What's the best way to assign ID's?
182. Why do you hate bad design so much?
183. Why do you hate old code so much?
184. What makes code crappy?
185. How much software is open source?
186. How far should automation go?

Chapter 8 - Software Business

187. Why start your own business?
188. I'm sooo confused...
189. The Investor Entrepreneur Chasm
190. How can I get started?
191. What went wrong in your first start-up?
192. Why are details so important?
193. Why are you writing your own software?
194. When do you say "yes"?
195. What drives development?
196. Why would you not launch?
197. Does consulting hurt a software start-up?
198. What do small business owners care about?
199. Product/Market Strategies
200. Differentiate or Die
201. Are the any advantages for single founders

202. How does it feel to be a single founder?
203. What should a business guy have to offer?
204. Where can I get help starting a business?
205. Why should you fire bad customers?
206. How important are ethics?
207. Why are ethics so important?
208. Do you conduct business over meals?
209. What's your favorite start-up book?

Chapter 9 - Enterprise Life

210. Willie Sutton would be an Enterprise Programmer
211. What do enterprise people need to know?
212. What is a typical enterprise day like?
213. What's is enterprise IT's biggest fear?
214. Why do excellent programmers leave enterprises?
215. Where are there real problems to solve?
216. How should layoffs be handled?
217. How do you test drive packaged software?
218. Have you ever been a hero at work?
219. Why is SQL so important in enterprises?
220. How did ERP get so screwed up?
221. Why is ERP becoming a dinosaur?
222. How tough is on-line retailing?
223. Why would we want to go faster?
224. When can code review be a problem?
225. Can business software be life critical?
226. How risky is free software?
227. What questions would you ask a new boss?

228. Why do enterprises stifle creativity?
229. Office Pet Peeves
230. User Pet Peeves
231. Do bosses lie?

Chapter 10 - Selling

232. How important is networking?
233. How do you find customers?
234. What do you talk about with prospects?
235. How should I handle a 1st customer meeting
236. How do you tell your story?
237. What Makes the Top 1%?
238. Buying Cycles
239. The Answer is Always "Yes"
240. How do you crack the enterprise world?
241. What are the Spending Authority Cut-Offs?
242. How do you close the deal?
243. How do I close a sale?
244. The One Excuse Not to Network

Chapter 11 - Just for Fun

245. What was your most creative resume?
246. Hacker News Front Page 12/31/2019
247. The Programmer's Aptitude Test
248. If Samuel L. Jackson were a Programmer
249. The Thread Not Traversed
250. Programmers are Practical

251. What if I miss a day of Hacker News?
252. Quality of HN Comments Over Time
253. Some Things Can't be Duplicated On-Line
254. A Time to Work and a Time to Play
255. A Programmer's Poem
256. Little Known Development Methods

Forward

Who am I?

My name is Ed Weissman and I've been programming professionally for 32 years.

I've done work for many companies, both enterprises and small/medium businesses. I've functioned as an employee, a contractor, and a vendor. I've worked in many industries, almost always on business systems.

I started out on IBM mainframes, moved to mini-computers, then to PCs, and finally to web-based technologies.

I've started three businesses, two with partners and one alone, selling both services and products.

I've worked with hundreds of people on over a thousand projects and encountered over a million lines of code.

I've witnessed a lot of business and technology, some great, some good, and some not so good. I've always loved what I do and can't imagine doing anything else.

I've formed lots of opinions and freely shared many of them on-line. So I recently decided to compile my favorite 256 Hacker News contributions here.

I never get too technical. There are plenty of other resources for that. Inside you'll find the opinions of someone who's been in the trenches for years and isn't afraid to say what he thinks. Whether or not you agree, understand, or even care, I hope you find something of value. That's the best I could hope for.

Why did I write this book?

Because you asked for it.

I started commenting on Hacker News when I discovered it four years ago. For a computer programmer, it had lots of cool articles, interesting commentary, and most of all, like minded souls. It became my on-line home - the location may have been virtual, but the results have been all too real.

Little did I know where this would lead...

Thanks for all the thought inducing discussion and encouraging feedback, both on-line and by email.

Lots of people claimed benefits from my comments and suggested that I do more, either by blogging or writing a book. My response was always, "I do write -- software." I was always afraid that devoting time to writing prose would take away time from writing source code. It took a while before I realized that this wasn't a zero-sum game; different kinds of writing can compliment and even enhance each other.

The feedback that finally got me over the hump was this comment from an anonymous poster:

"Seriously, have you ever thought about writing a book? I learn something new and valuable every time I read a comment of yours, since your comments almost always contain really good tips. I for one would buy that book for sure."

(Thanks, Mom.)

My comments have visited far more interesting places than I ever have:

"Harvard Business Review"

"mixergy" by Andrew Warner

"Hacker Monthly"

"SKMurphy" by Sean Murphy

"JasonLBaptiste"

"allExceptTech" by Ketan Khairnar

"FundFindr" by Bret Conkin

"University of Washington Orthopaedics and Sports Medicine" by Jonathan Jacky

"Why does everything suck?" by Hank Williams

"Sixteenth Letter" by Melissa Chang

"Back of the Envelope" by Mr. E

"Soba Soup"

"omelettesoft"

And finally, some of the nice things people have said about the comments of my on-line persona:

"...edw519, most of his stuff is thoroughly thought-provoking..." - Matt Mazur

"The Best Comment Ever (No, I Didn't Write It)... by edw519, writing an incredible comment on an otherwise unworthy Hacker News story." - Mark O'Connor

"...another great comment from edw519..." - Sigit Nurseto

"...an excellent follow-up comment at HN by uber-poster edw519..." - D. C. Toedt

"...Ed Weissman (edw519 on HN) had another great comment recently on Hacker News...and he offers a great list of reasons why..." - Sean Murphy

"From Hacker News commenter edw519 comes this gem..." - Andrew R. Freed

"...Hacker News reader edw519 makes a great observation..." by Rams

"And then there are the folks who have such an intelligent and unique voice that you can tell who the comment is before you see their username (patio11 and edw519, to name two)." - Ryan Waggoner

Thanks to all of you. I hope to pass on your positive energy to others in some way. This book is my first step.

How did I write this book?

I built this book the way any self-respecting programmer would: with lots of shortcuts and software.

Notice I said "built", not "wrote". The contents were already written, as contributions to Hacker News, a large on-line community for computer programmers.

What I did:

First, I scraped the interwebs for every comment I ever made on Hacker News and put all 4,000 of them into a hashed database on my c: drive.

Then, I converted the text into something more presentable, removing formatting, control characters, and cuss words. For example, whenever you see "crap" in this book, you can bet that it was something much "stronger" in the original.

Finally, I wrote a program to turn those comments into a usable book. Some of the things that program does:

1. It displays the comments, one at a time or in groups, enabling me to cruise through them very quickly, using only arrow keys.

2. It sorts the comments five ways: by descending number of words, by descending points, by descending date, by descending weight (a secret formula), and in output sequence.

3. It enables me to quickly classify any comment by suitability, chapter, and sequence.

4. It filters the comments by classification.

5. It enables me to drop down and edit any comment.

6. It automatically classifies unread comments based upon similarity to classified comments and some rules. (The idea was to classify the first 300 comments and have the software classify the remaining 3,700. I realized this capability was unnecessary when the book would only contain 256 entries. Oh well.)

7. It cuts the book in multiple formats, ready for distribution.

Like a typical programmer, I often got confused about what the real product was. I spent more time on the software than the contents of the book. Then I decided to get the book out and finish the software later. I intend to make it available on-line at some point so that others can assemble their own books.

Chapter 1

Advice to Young Programmers

1. What's it like to be a programmer?

"1. What are some qualifications of a computer programmer?"

The two most important qualifications are a love of details and a simultaneous appreciation of the bigger picture. You have to understand the landscape that your software will fit into. Then you have to be willing and able to dig down deep and be comfortable building stuff at the lowest level of detail. This takes a great deal of logical thinking, attention to detail, and personal focus.

"2. What is the best part of being a computer programmer? The worst? The most challenging?"

The best part is getting something working for the first time where nothing was there before. For me, this is so exciting that I still I do a "happy dance" every time. The worst part is the long hours alone. There's really no way around it; good software takes time and almost everything is done by someone alone at a terminal. The most challenging is finding a project big enough to not be boring but small enough that's it's too difficult to make good progress.

"3. What's the salary range in this career?"

As an employee, $35,000 to $200,000. As a company owner, $0 to billions. Either way, the range is very wide and depends on many factors, some outside of your control. Like any other profession, you should be a programmer because you love to program, not because of how much money you'll make.

"4. What is a typical day in the life of a computer programmer?"

I bet there are as many typical days as there are programmers, so I'll just share mine. My day starts at my terminal, making changes to my current program based the mark-ups I did to my hard copy in bed the night before. I spend most of the day at the terminal writing code, changing it, trying it out, and taking occasional notes. I avoid interruptions as much as I can. I have a regular lunch and dinner and some social life, but not too much. Every day

ends the same, in bed with whatever I worked on that day, reviewing and marking up. Incredible attention to detail is required and this is how I do it.

"5. What is some advice you would give to young computer programmers?"

Just build something. Nothing can be more important. Whenever you need to learn something, find a way to learn it, whether it's a class, friends, or more likely, a book or website. It you want to be a programmer badly enough, you'll find this approach natural. If you don't, you won't.

"6. Is it easy to find a job as a computer programmer?"

If you're good (and can prove it), yes. It not, not so much.

"7. What was your most exciting project?"

A computer program that wrote other computer programs.

"8. What skills do you think young programmers need for the job?"

The ability to think clearly and logically, good written and verbal communication skills, the discipline to keep working when they'd rather be with other people, and the determination to see something through to completion.

"9. What improvement does computer programming give for human life?"

Computer programming makes software that frees people up to think about and do things that weren't possible just a few years ago. The possibilites for those people are endless.

"10. What is the future direction of computer programming?"

This is always hard to predict, but I'd guess the direction will head away from writing all of your own software toward

connecting a lot of already written software to accomplish the same thing.

"11. Would life be a lot worse without computer programming? How much? Why?"

Just compare life in a country with advanced technology to one without. Computer programming doesn't have everything to do with the difference, but it does have a lot. Much of today's advanced lifestyle has resulted from modern technology. Much modern technology came from software. All software came from computer programming.

2. How can I get started in programming?

Oh how I wish I could share the joys of hacking with non-hackers, but that would be like describing the color blue to a blind man. You just gotta experience it yourself. There's nothing like putting something together and seeing it work the first time. Even if it isn't perfect, that first output is better than sex. Still makes make holler and jump out of my chair! (The output, not the sex.)

I would strongly suggest trying out one of the many "Build an App is x Days or Hours". Grab a book or something on-line. They're everywhere. Follow the instructions and do what they say. Build you app.

One of two things will happen: you'll either feel like I do and you'll be hooked. Or not. Either way is OK, but to not give it a shot this year would be a shame.

(By the way, you'll probably find out that's how the best of us got started. School is cool, but nothing replaces just doing it.)

3. How should I learn web programming?

"I want to code the site using PHP and a bit of Javascript, but my skills with these are not exactly up to the job yet."

You must understand that you need to learn 2 separate things and you need to learn them well.

For javascript on the client you need nothing other than the browser you already have and the Rhino book:

http://www.amazon.com/JavaScript-Definitive-Guide-David-Flan...

Learn what's in this book! Go through all the exercises and tutorials. Build something. You can augment the book with tutorials you find on-line. Then you can View Source on any web page and understand what they did (and what they did wrong).

On the server you will have to find any common LAMP stack and load in onto your machine. The execises and tutorials for php, MySQL, and apache should be enough, although you can find more almost anywhere. Build something! Now that you already know javascript, you can include that in the pages you build as required.

Only after you have a solid understanding of the basics of these 2 technologies should you consider a framework. This can be tricky. If you adopt a framework too soon, you may run into a problem for which you don't understand enough about what's going on under the hood because you never learned it. If you adopt a framework too late, you'll be hand coding everything and will never get done.

Most importantly: you can only learn any of this by doing. Time consuming doing. Books and resources any necessary but hardly sufficient.

Do not fall into the trap of only learning at the surface and expecting to find someone else to do the coding. This does not work for a small software start-up. You must dig deep and learn well.

4. Should I try programming if I'm not sure?

I'm not going to try to motivate you, because only you can motivate yourself. I'm just going to share my perspective that might shed some light on your issue.

I had the same problem in college until a fraternity brother who had graduated 2 years before told me something I had never thought of. He said, "You may never have a better opportunity to explore and learn new things. Once you settle down with a career and family, all your time will be spoken for. So try everything! How will you know who you are and what you're interested in unless you experiment?"

Some of the best advice I ever got, and since I was a sophomore at the time, I tried as much as I could for the next 2 1/2 years. I still majored in math and became a computer programmer, but I did a lot of stuff that I simply don't have time for today. And I miss a lot of it. Back then, I thought humanities was boring, but what I would give for a few days off to curl up with a good book.

Sometimes you can learn a lot about something you are interested in by doing something totally different. Reading literature might help you write code. Music may help you with math. Working at the mall or living in a fraternity house could help you learn how to run a business.

At this point in your life, you still don't know (for sure) what you'll love and be good at for the rest of your life. So keep doing those things that "seem" like they're boring and uninteresting. Many of them will be a waste of time, but there will certainly be a few pearls in there, too. Don't miss the opportunity of discovering them because you just want to stay in your comfort zone. Now is the best time to leave that comport zone and discover the other stuff. Eventually, you'll be glad you did.

5. How do I find passion for my work?

The reason you're not passionate about your work is because something is missing. Identifying what is missing is your first step in determining where to go from here.

I have been in a similar situation. Always working. Important stuff. Sometimes cool, often not. But something was always missing.

Architecture not rigorous enough. Inadequate data base design. Insufficient requirements definition. Lousy code base. Unable to scale. Unable to expand or handle completely new features.

But I always managed to make it work anyway.

Then it occurred to me, if such mediocre systems were able to produce adequate results in commercial environments, what would be possible with great systems?

So now I'm building a framework/architecture/environment that beautifully handles everything I thought was missing before. The passion is built-in. Instead of, "Look at me, ma!" now it's "Look at this, everybody!"

Where do you go from here? Fill in the gaps that should have been providing passion all along. That oughta keep you busy for a while.

6. Is this a good way to get motivated?

"There's a way for me to make some money, but it requires that I setup a fairly complicated spreadsheet to monitor several variables."

A "complicated spreadsheet" isn't a requirement, it's a roadblock.

You're making it harder for yourself by putting obstacles in front of yourself and then wondering why it's so hard to make progress.

"I want to be rich. Filthy rich, even."

Getting rich isn't the goal. It's a byproduct.

You never even mention what your startup is going to do, who it's going to help, or why you absolutely positively must do it. If you have something you "must" do, identify it and focus on it. If you don't, find it. Everything else, including money, is just a detail.

"I take a break to 'clear my head'."

Clearing your head isn't a necessary step, it's an excuse. Again, if you have something you "must" do, you head is already plenty clear. If you don't, then what are you clearing your head for?

In summary:

1. Find what you "must" do.

2. Start doing it.

In case you don't have something you "must" do, then just do something, anything. The process of doing will probably help you find your mission. The processes of thinking, preparing tools, and dreaming about money probably won't.

A couple of minor pointers that have helped me:

1. If you have 2 computers, make one for work and the other for internet and "put them in different rooms".

2. Throw your TV set into the dumpster.

3. When you have code to work on, be at your terminal, working on it (Mode 1).

4. When you don't have code to write, be anywhere but your terminal with pencil and paper handy (Mode 2).

5. Start every day in Mode 1 and end every day (probably in bed) in Mode 2. Ending the day in Mode 2 in requisite to being able to start in Mode 1 the next day.

6. Take care of yourself.

7. A Computer Scientist who doesn't want to Code

"Am I in the wrong major?"

Yes.

You (and some others) may not like what I'm about to say, but you asked for it, so here goes...

In all the years I've been in technology, it has typically taken me about 28 seconds to determine if another person was "fluent" more than one or two levels below the surface.

Those that were were almost always programmers, engineers, or technicians at one time or another. Everyone else was at best managers and business people, or at worst, administrators or posers.

I know some might disagree with me, but a Computer Science major who doesn't want to code is like a dental student who doesn't want to look into anyone's mouth.

To get good in technology, and I mean really good, you must get under the hood, deeply and often. The best and most logical way to do this is by programming. And you will have to do this intensely and for long hours, so "you have to love it".

The single biggest difference I've seen between great programmers and everyone else is a pure love for what they do. Intelligence matters, work habits matter, ability to work with other people matters, but make no mistake about it, there is no substitute for passion.

Great technologists love what they do so much, they can't wait to get back to it. They have to check on their work after dinner. They have to review their notes at bed time. They are often the first in the office in the morning and just as often the last to leave. They read and learn voraciously and can't wait to apply their skills to new problems. They're so busy doing what they love, they don't even think of it as "working 9 to 5".

By your own description, you do not sound like this. So do yourself (and the rest of us) a favor and find something you love and major in that. If, on the other hand, it's too late or it doesn't make sense to switch majors, then go ahead and finish your CS major, but please find a direction to follow that puts you in work you love. Be forewarned, though. Unless you're a programmer first, you probably won't make a very good sales engineer or project manager. You may want to consider sales or even (dare I say) proceeding on to business school for your MBA.

8. What should a mediocre programmer do?

"I'm really just a mediocre programmer with really good domain knowledge"

Your problem is that you have no problem. Let me explain...

I believe that the quality of a programmer is not how much you know, but what you can do with it. So if you have "really good domain knowledge", then you probably aren't a mediocre programmer at all, you're probably a good programmer or even better.

Like many other programmers, I love to check out the latest cool stuff people are doing. Then I hear the 2 voices in my head. One says, "That is so cool - I have to learn that!" The other says, "Big deal, I could do that in BASIC. I may need a few more lines of codes and a couple of hacks, but it will still do the exact same thing."

It's tricky to balance all the cool stuff going on with your ability to "just get stuff done". You will never learn everything. You will never become the expert at more than one or two things. It's great to "learn", but not at the expense of "doing". You need both. There were many times I had to build something with my limited knowledge and wished I knew more. But then I built it anyway. Something built with limited resources today is better than something built perfectly tomorrow.

If you're unhappy with your job but like coding, then either find another job or start something on the side. But please don't fall into the trap that you aren't good enough because "someone else" knows "something more". That will always be the case. You can't win that battle.

Just do the best with what you have and make a practice of adding to it a little at a time. Get satisfaction from the benefits you provide others with what you know now.

9. Am I burnt out?

You are not burnt out and I have proof. This discussion. A truly burnt out person would not have even bothered. (Kinda like claiming you're over your ex-girlfriend but still wonder what she's doing all day long). The fact that you asked is not an admission of giving up; it is a cry for help. You still really want this.

I go through what you are experiencing all the time. There are days when I can't stay awake at my terminal. Sometimes I hit a road block and wonder how I'll ever get by. I usually step away for a time, but here is my real secret...

Pick one little thing that needs to get done, no matter how small or unimportant it may seem. If I'm really down, I pick some mundane task like refactoring 25 lines of code, manually updating 50 records, or even changing some naming conventions. But not something big like solving a client-server architecture problem. Hell, that's the reason I'm already down. One other thing - the task must be in the heart of your project; cleaning off your desk or reading a journal don't count. Then do the task. Completely. You'll feel a little better, I promise. The next day, do it again, maybe with a slightly bigger task. And again. And again. Who knows, you may be feeling a lot better before you know it.

I have no idea if my advice can help save your project, but I do know you still want to. Use others for support. You are not alone.

10. How can I find ideas?

Here's an idea: get a job. After a year, you'll have plenty of ideas, maybe even one of your own.

I hate to rain on anyone's parade, but the thought of begging for ideas is just so alien to me. The best predictor of your success in any endeavor is your own determination. With someone else's idea, you're much more likely to bail at the first sign of difficulty. Once you get a little real world experience under your belt, you'll find plenty of opportunities to encounter something for which you'll have real passion.

Your chances of success increase astronomically when you're working on something you "have to do". The only way to know if you "have to do it" is to have a little background and experience with it. Trading ideas like commodities seems like the least likely way to find something you'll be passionate about.

OTOH, a "boring job" can be an incredibly fertile environment for start-up ideas. You'll learn what people want, see what works and what doesn't, and be much more adept at identifying opportunities. Oh, and get a chance to bank some money so that when you do start working on your passion, you can concentrate on that instead of begging for funding.

Sometimes the easy way out is just that: the easy way "out". Get a job and pay your dues. You'll probably be glad that you did.

11. Should a systems analyst code?

I hate to break the news to you, but "systems analyst" is a job title that's pretty much exclusive to the enterprise world. For the most part, start-ups don't really have a place for "business/system analysts". Let me explain...

School of Thought A: Call it SDLC (Systems Development Life Cycle), the project approach, or the waterfall approach goes something like this: Define a need, conduct analysis to answer the question "What", conduct design to answer the question "How", program, test, program, test, compare to the Functional Specs from the Analysis Phase, conduct User Acceptance Testing, promote, deploy, repeat anything as required. The more rigorous the documentation and project management, the better.

School of Thought B: Build something ASAP. Get it out there ASAP. Get feedback ASAP. Iterate indefinitely.

For the most part (I'm sure there are many counter-examples), enterprises employ School of Thought A and Start-Ups employ School of Thought B. There simply isn't a need for systems analysis in School of Thought B. By the time you're done analyzing, someone else is servicing the customers you sought.

My advice: Combine a love for building stuff with your love of systems analysis. They go perfectly together. In fact, we now have a name for that: "programmer/analyst".

The systems analyst who can code is a better systems analyst because he can test/evaluate his ideas.

The programmer who can conduct analysis is a better programmer because he knows what to work on.

This may not be what you wanted to hear, but you're in a perfect position to do both, so do it.

12. What's the best way to find ideas?

We are constantly hearing advice like "Scratch your own itch," "Find problems close to you," and "Code what you know," which is all good advice for finding a startup idea.

What then, do you do if you're not exposed to much?

When I spent time talking to people in Silicon Valley last year, I noticed something I never expected: lots of people have very little exposure to "real world" problems.

People come up with "me too" startups because that's all they know.

Sure it's easy and cheap to start a business, and if you're smart and can hack, all the more reason. But what about the giant issue hardly anyone mentions: real world experience.

I've been doing programming work in real businesses for years and "still" get excellent startup ideas from my customers almost every day. In the past week alone, I've been challenged with problems I've never seen before that are really affecting these people. Just a couple of quick recent examples:

- One customer is preparing blanket purchase orders for new models, but since the SKUs change every season, their ERP system is of little help.

- Another is setting up a new warehouse, but their app won't allow bulk updates so they have to hire clerks just to enter 10,000 new bin locations.

- Another has a call center whose response time has doubled with an upgrade to their app. Now they want the new release's functionality with the old release's interface.

And this is just one week! People will pay big bucks for technical solutions to these real world business problems.

My advice to a young hacker with lots of skill but not much experience (outside of college apps): you may want to consider getting a job for a year or two. I know, everyone wants to start the next , and get rich now and no one wants to sit in Megacorp's cube farm. And a startup, even a failed one, may be a better education.

On the other hand, you will acquire dozens of great startup ideas (and contacts) and avoid the "me too" trap. Something to think about.

13. How do you become a fearless programmer?

I have always been a "fearless" programmer, but never realized it until recently. Here's how:

"Fear of not knowing the best way to do things (best practices)."

The sooner you realize that there is never a best way of doing anything, the sooner you can release this silly fear. Some ways are better that others, but "any way" is better than "no way". Just get the thing done. Later, when you refactor, you'll have the best of all worlds: code that did the job right away, a better way of doing things, a satisfied customer, and a great learning experience.

"Fear of not using the right tools and languages."

Give me an adjustable wrench, 2 screwdrivers, and a big hammer and I can fix just about anything. Same thing with programming. I'm too busy getting work done to learn every new tool or technique. As I've told many programmers over the years: "Whatever you can do, I can do in BASIC. Maybe not as pretty, but probably just as fast and just as effective."

"Fear of errors (especially compiler errors)."

You're in the wrong business. Errors are what point you in the right direction. The sooner you learn to embrace errors and use them to refine your work, the sooner you'll become fearless (and better).

"Fear of schedules."

"I see only one move ahead, but it is always the correct one." - chess master Jose Raul Capablanca. That's what my schedule looks like. One item. One day. Project managers can't stand this, but then again, I get way more work done than they do.

"Fear of publicity (what will other programmers think about this code?)."

I never publish my code. Ever. Users get to give me feedback, but I don't care what other programmers think. Sure, I learn from them, but never in the context of reviewing the code I wrote. I learn from the code of others and apply those lessons to my own work.

14. Should I still be a programmer?

"I lack the fundamentals of Computer Science, the things every programmer should know: Algo's, Data Structures, Operating Systems an understanding of compilers and being profficient with linux."

Relax. That's true for 99% of all programmers.

"Eventually I plan on going back to a real University and getting a CS degree"

Absolutely not necessary. You will probably learn more building anything than learning it in school.

"I'm starting to think I may have a learning dissability"

Maybe you do, maybe you don't. Just because the rest of the world is quick to diagnose everything doesn't mean you have to.

"I have come to accept that I'm really not smart. I'm slow, forgetful, concepts never seem to stick"

Your performance shortcomings could be for many reasons. Being "really not smart" is the least likely of any of them. They are much more likely caused by other things like uninteresting work, poor environment, personal issues, nasty people, or even health issues. Whether you're "smart" or not, thinking that your aren't is pretty much a guarantee for failure. Please don't do that.

"I have started going through the basic Algo's and Data structures again with a basic Java book about algorithms"

Sorry to say, but you're doing it the hardest way you can. You don't need a book; you need a project. I have achieved much, but have always had difficulty learning from books and theory. It's hard! You need to find work where the things you need to learn will be required. Funny how quickly and easily you'll find a way to learn them when you actually need them. I'm not sure how you should go about finding such work, but I know you'll figure that out.

"I just don't want to be a cargo cult programmer anymore."

Good. That makes your normal.

"And finally trying to memorise all those linux commands I ALWAYS forget."

Then use less commands. I have never used more that 10% of what was available in "any" technology and I always got the job done.

"I have to force myself"

This is the most important thing you have said.

Have you ever actually enjoyed building stuff? Have you ever gotten really jazzed about the project you were working on? Have you ever lept up out of your seat and danced when you got something working?

If you answered "yes" to any of these questions, then you really do have the passion to be a programmer. Stop selling yourself short.

If you answered "yes" to none of these questions, then there's no sense for you to continue wasting your time searching for the passion. If you haven't experienced any by now, then you probably never will. No one should have to "force themself" to love what they're doing. Find something else.

15. What if my problem has many competitors?

Here's an idea: find "someone else" with a problem and work on that. This works especially well if the someone else is in business, very busy, and has some money. Chances are better that your solution to their problem won't have much competition: if it did, they would have already gone with it.

I know this is the opposite of "scratch your own itch", but I always found that advice overrated. I have always been much more successful scratching other people's itches.

16. What would you advise your younger self?

Find a customer first.

This is the biggest mistake I made in my twenties, and it's such an easy mistake to make that I continue to make it now even though I know better.

I continually have ideas popping into my head. And I act on many of them. So much cool stuff. If only I can get this working, it will change the world. And I love being in this mode; it's so much fun. And it can lead to great things...

But you have to know when you're going too far and wasting time, money, and energy. At some point, you have to find a customer. Any kind of customer, just someone besides yourself who wants what you're doing.

When I have had partners, they forced me into this thinking, directing our energy to where the demand was. This always worked out well.

When I worked alone, well let's just say there's tons of cool stuff still on the drawing board that led nowhere. Don't let that happen to you.

I'm not saying to suppress your creativity or experimentation. I'm just saying the point you need to find a customer is much earlier than the hacker mindset intuitively expects.

It there was one thing I could change in my twenties, it would be to adopt this thinking.

17. What was your startup's biggest mistake?

I wrote a code generation system that put together nice tight little apps, with UI, database interface, batch processing, and a report generator. It came in really handy for the simple apps everyone seems to need every once in a while.

My partner and I sold a few simple apps and then he found an opportunity in a large company for a much more sophisticated app. He quoted based upon the already demonstrated performance of me and my "little tool".

Unfortunately, every single unusual thing we ran into was not "generatable" by the tool. So I had a choice, hand code or upgrade the tool to generate it.

My choice was my "best failure story". I chose to add funtionality to the tool to generate code needed for an open project already well behind schedule. I didn't stand a chance. Even hard coding everything probably would not have saved the gig.

Lessons learned:

- Code generators must have hooks for custom code.

- Don't commit to using a code generator until after you have the customer's requirements and know that the generator can already handle them.

- Don't build your tools on the job site unless you're really good and know that you can finish on time.

- Don't let a few early and easy successes let you get a big head. It's never as good as it seems. (It's also rarely as bad as it seems.)

- Have complete, open, and honest communication with your partner(s). You don't have to know every single thing

each is doing, but you better be on the same wavelength.

- Put everything dealing with external parties (customers) in writing. Commit to nothing until everyone agrees.

- If there's only 2 of you, you both better be hackers. There's just too much technical work for one to be overwhelmed and the other to be "waiting".

18. Should I leave a job I love?

"the job happens to be my all-time favorite"

What are you crazy?!?

I've had many jobs and I've liked few of them. If you have a job you love, why would you want to leave? You don't even have any plans. I'd understand this a little better if you had a project you're dying to work on full time, but that's not the case.

Why don't you just keep your job and find a side project. If that side project gets big, go part time. If it gets so big, you're "burning" to work on it full-time, then quit, but not before.

Good jobs are hard to come by and jobs you love are almost impossible to come by. Also, don't discount all the data you get from your job to feed your startup plans. Lots of people would love to do a startup, but don't know what to work on. People with jobs don't have that problem as much. The job can be the source of lots of great ideas for things people actually need right now.

I'm the last person to discourage anyone from doing their thing, but job vs. startup is not a binary decision. You can do both, at least for a while.

Keep that all-time favorite job for now. You can always leave later once something else has wings.

19. Planting Seeds or Reaping Harvest?

"I took a step back and immediately saw absolutely no redeeming value of what I had just done, and this made me feel incredibly sad."

I would rephrase that as, "...absolutely no redeeming value "at this time"..."

You have just planted seeds. Of course you can see no "redeeming value" right now. If you just planted seeds in your garden out back, you wouldn't see any results there either, "today".

But just as the seeds in your garden will deliver results in a few months (provided you care for them), the seeds you just planted in your little "hairbrained" project, will also reap dividends. You just don't know when or what.

Sometime in the future when you least expect it, the lightbulb will go on and you'll figure out a cool solution to some other problem. What you don't know now (and probably won't even realize then) was that what you did today stirred a few neurons to enable that light bulb to go on in the future.

As hackers, "everything" we do is either planting seeds or reaping harvest. If you didn't see a harvest today, that's because you were planting seeds. Just keep on working and trust the process; that's how we all get better at what we do.

20. Have I wasted 3 years working for someone else?

Wow! You have had what seems like an ideal first three years after college and all you see are negatives. Have you learned anything in that time? From what you've described, I would think plenty.

The last 3 years was a process you had to go through to get to where you eventually want to be. A very efficient process that not too many others ever get a chance for.

Every time I had a tough gig, instead of focusing of the jerk I worked for, all I thought about was what I was learning and how I could leverage that into my own future. That kind of thinking has always worked well for me.

You now have the background, experience, and skill of an accomplished 35 year old in a 25 year old body. Congratulations!

Now stop worrying about the past, take a short break, and put those hard earned assets between your ears to work for what you really want. This time next year, I hope to hear from you, "How I turned 3 years of sweat into a lifetime of fulfillment."

Chapter 2

Education

21. How much formal education do I need?

"Would my money be better spent on a MBA or MSCS?"

Neither.

1. Find someone who needs something.

2. Start building it.

3. Trust that when you need to learn something, you will.

The education you will receive this way will be way better than any formal education for multiple reasons. First, you will automatically triage your lessons; you will learn what you need, not what someone else (who probably doesn't know) thinks you need. Second, almost all the "data" you will need in this education is easily available and free. Third, for the education you need from other people, you will begin building a network you'll need anyway. And finally, this is exactly what you'll have to do "whether or not you get any more formal education", so just skip the unnecessary step and get on with on. From your own self description, you already have way more formal education than you need.

This may not seem intuitive, but believe me, this is the way things get done in the real world of software development. At this point, the cream rises to the challenge regardless of education. Save your money for living expenses and start-up expenses. You'll probably need it. Best wishes!

22. How important is my degree?

As smart people who deal a lot with binaryness (there "is" a right answer), we place high regard for education, as we should. But please understand that business and academia are 2 different animals with only a little overlap. I have lots of education and lots of practical work experience, and I have to be careful "when" to apply my formal education, which isn't often.

Classic example (I'm sure many of you have many more):

I graduated Allegheny College with the founders of ijet.com. At an alumni event, they presented their business case and web site. It was very interesting. At the end, a business professor summed things up. Not a single thing he said made any sense or had anything to do with the presentation. I wondered if he had even watched it. That's when I decided to stop feeling bad about sleeping through all that Cost/Value Curve bullshit so many years ago.

Get your degree. Get your education. But please understand they aren't necessarily the same thing.

23. What should I do in college?

```
Take science to discover something you're good at.
Take humanities to discover something you may love.
Take at least one art or music class.
Take at least one advanced math class.
Join a fraternity.
Learn how to play bridge (and play all night sometime).
Learn how to play foosball.
Get drunk.
Learn how to play foosball while drunk.
Play an intramural team sport.
Get a part time job.
Eat something you never tried before at least once/month.
Do original research.
Take a class you think you'll hate pass/fail.
Do 5 minutes at a comedy club on open mike night.
Hang out with a professor you like.
Do a web start-up on the side.
Make a few friends for life.
Go to at least one party each week.
Pick a major you love whether it makes career sense or not.
Get someone who has written one of your text books to sign it.
Blog about your college experience.
Go to Europe with nothing but a backpack for a month or two.
Enter a college talent show.
Meet as many interesting (and boring) people as you can.
Read good books.
Go without shoes for a week just for the hell of it.
Get laid.
Graduate.
```

If you don't go to college, exactly when do you expect to do all of this?

24. What did you learn in school?

You can find plenty of important life skills to be learned in school if you only look hard enough:

```
kindergarden    - learn how to play nicely together
first grade     - learn how to read and write
second grade    - learn how to add, subtract, multiply, & divide
third grade     - learn how to spell
fourth grade    - learn how to play a musical instrument
fifth grade     - learn how to appreciate great literature
sixth grade     - learn how we got where we are
seventh grade   - learn a foreign language
eighth grade    - learn how to type and use a computer
ninth grade     - learn how the world is put together
tenth grade     - learn about other people in the world
eleventh grade  - learn how to balance a job and school
twelveth grade  - learn how to plan for and dream about the future
freshman year   - learn how many other kinds of people are out there
sophomore year  - learn how to chug a beer, fill a bong, and get laid
junior year     - learn how to stand upon the shoulders of giants
senior year     - learn how to find your place in the world
graduate school - learn how to play nicely together, all over again
```

25. Should a programmer get an MBA?

I honestly can't think of any good reason today why someone who could hack would want an MBA.

If you're concerned about learning, you will learn more about business in one good hands on project than 2 years of formal education.

If you're concerned about having a mentor, you will meet much more business saavy people in business than you ever will in school.

If you're concerned about money, 2 years of earning will put you hundreds of thousands of dollars ahead of 2 years of spending.

If you're concerned about what others think of your credentials, you're focusing on the wrong thing.

If you're concerned about climbing the corporate ladder more quickly, then fine. Get an MBA.

And, most of all...

If you're concerned about making yourself the best you could be, then hack, hack, hack. You already excel at the weakest link in the business chain. You will get better by continuing to "do" in your specialty. More school will add little, cost you time, cost you money, but most of all, cost you experience doing what you already love and what the world needs most right now.

26. Why don't you think an MBA is important?

"all the things that you would need to learn in a full-time 2 year MBA program"

I can't think of a single thing that would be on that list. Business School is not like Law School or Medical School where you must remember the "things" you learned. Employers use the MBA to differentiate candidates. It's unlikely that you'll use all that much from the curriculum on your first job. You even say so yourself, "firms hire MBAs on their abilities to learn and not what they've already learned".

"I just don't see how that is remotely possible."

How could you if you've never done it? Every large project I've ever worked on had issues with interpersonal communication, project management, logistics, deployment, revenue generation, and profitability (you know, all the important stuff), in a manner that business school can't even imagine covering.

"the problem is the quality not the quantity"

Exactly. That's my whole point. The quality of a hands on business education blows away anything academic. I imagine most business people (with MBAs or not) would heartily concur. The only thing I remember from business school was, "A degree in business is a degree in nothing."

"in-depth case analysis"

Case studies are notoriously poor for learning about business. What good is it to study business decisions after the fact, when you already know what they didn't? If you don't want to listen to me, perhaps he's a little more convincing. (Whether you agree with me about anything or not, I strongly suggest this video. A lot to learn from someone with real battle scars.)

"with your professor and the other students"

who pale in comparison to people your encounter every day on the job. Why do you think the most important class in any business school is the internship?

27. How important is a degree in business?

"Instead of students studying Literature, Art, History, and Science they would be going through the motions of a scholar while occupying their minds with things that formerly had been learned at a desk as an apprentice in a dreary Victorian counting house."

That may be the best description of the concerns of the MBA I've ever read.

I have my MBA and, to this day, I still don't know how I feel about it. Sure, it covered a lot of valuable theory and it's opened doors, but then again I often wonder if the time would have been better spent in industry, honing my skills in the trenches.

28. What do you best remember from your MBA?

I understand OP's sentiment. I place very high value on my MBA from the University of Pittsburgh, but for reasons one wouldn't expect.

I remember almost nothing from the course material itself. You can get that from almost any book. The time value of money, how to read financial statements, the 4 P's of marketing (or was it the 4 B's?), different management theories, etc., etc., etc.

What I remember vividly are the interactions with other people, professors and other students, some of whom already had many years of experience. I'm not sure how else I could have had this experience. Some of my most vivid memories:

"A degree in business is a degree in nothing." - management professor

"What is the correct answer to the question 'How much money did you make?' It's always, 'Who wants to know'?" - managerial accounting professor

A professor of Organizational Behavior told us that a survey of Yale MBA's 25 years after graduation said that Organizational Behavior was their most important course. I couldn't believe it. Years later, I believe it.

A case study had our team pick the next CEO from 4 managers. We were wrong. The actual result: They went outside, for reasons well explained.

Our group made a recommendation to a local business to restructure, letting a key person go. We got an "F" on the project, with the opportunity to go back and redo it without letting anyone go. We got the message.

A marketing professor was picking on stupid products, like Heinz gravy, when a student in the back offered, "I was the product manager for gravy at Heinz. It made us $60 million last quarter. If you'd like to use it for a case study

some day, I'd be glad to help you with it."

I don't remember many specifics from class, but I know that I "think differently" because of that experience. Along with a technical background, this has been a great combination.

29. What happens at Toastmasters

My experience (other chapters may differ):

1. Everyone who comes speaks to the entire group for 1 or 2 minutes. This was called "Table Topics". A different member is assigned each week to administer the Table Topics. I remember one time the Table Topics person handed each one of us a paper bag that we had to open, remove the object inside, and give an interesting talk. My object was a clothes pin.

2. Everyone gets a mentor when they join to help them through the education.

3. Everyone cycles through the program at their own pace giving longer and longer prepared talks. 5:00, 10:00, 15:00, etc. You give one of these every month or so.

4. Every single talk is publicly critiqued by at least one judge. They are usually ruthless and very, very helpful. It took me months to stop playing with my hands when I talked. They finally got me to stop.

5. I remembered speaking about my business in all my prepared talks. I can't imagine having a better place to practice than Toastmasters.

30. Why should an MBA learn programming?

"You will NOT become an engineer, programmer, or web developer, but you will be able to put a prototype of your idea together and maybe get one or two beta users for feedback..."

You will also be able to have an intelligent conversation with a developer.

I get sad whenever I encounter a business person with no technical BS filter. Not because I'm judging them, but because if I can BS them, any other developer can. Which probably means there's a problem somewhere that will hurt all of us.

I'll make a point to have a cursory understanding of financial statements, market segmentation, and project management if you do the same for the basic building blocks of software applications. Then the two of us will be able to talk about almost anything. OK?

Chapter 3

Careers

31. What are employers looking for?

When I hire, I'm not looking for a person or a resource. I'm looking for a solution to my problem. Sometimes that problem is big, sometimes it's urgent. But there's always a problem needing to be solved. The more a candidate looks like a solution to my problem, the closer to the front of the pile he/she gets.

As far as I'm concerned, the most important thing for any candidate to do is to identify my problem(s) and present themselves as the solution. The problem could be:

- We just got a bunch of new business and need someone to do immediately to satisfy those customers.

- We just acquired another business and need to convert/integrate from technology to technology and need someone who knows both technologies (or either one) and has done that before.

- We have a new business problem and one possible solution is to build/enhance/maintain an app. Can you do that? Have you done that?

- We plan to grow x% over the next 24 months and we need people to do more of what we already do which is . Can you do that? Have you done that?

- We have a problem and frankly, we're not sure what to do. What would you suggest? Can you do that? Have you done that before?

You kinda get the picture. The tricky part for any candidate is the research. How do you find out what my problems are? Ask! Ask me. Ask someone else in the company. Ask anyone. The simple act of research shows that you're a serious candidate. The follow up with a solution to my problem puts you at the top of my list.

If you're right out of school or don't have a lot of experience, you should still do this. You may not have as long a resume as others, but you have every bit as much to offer to solve my problems. Maybe a smart person who works

hard and knows how to deliver is just what I need. You must find that out and present yourself as such. Remember, it's about my problem, not yours.

This was I great question. I've never seen it before. The fact that you thought enough to ask is a "huge" first step. It shows that you're thinking about me, not just yourself. Keep thinking like that and there's no telling how far you'll get. Thank you and good luck.

32. How do you prepare your resume?

"Do you prefer a single page resume or multi-page? If multi, then how many pages of resume you think is good enough to sell you?"

Single page. If I absolutely, positively have more to say, I occasionally attach a one page Appendix, "Sample of Project Particulars," which includes 5 or 6 quick stories about major projects I completed that are relevant to the company and position I'm submitting to.

"Do you elaborate on your work experience (like, job description, responsibilities, etc.) or you want to keep it short?"

Yes, but I wouldn't say "elaborate". More like "itemize". Forget about things like "experience", "job description", or "responsibilities". Focus on one thing only: results. What I did, who it was for, why they needed it, and what they accomplished with it. "Built an AJAX e-commerce site that enabled a $10 million catalog distributor to double sales in 6 months." Show that I understand the forest in which I'm planting trees. Short, sweet, and to the point. If it catches their attention, they'll ask you more about it. If it doesn't, then you probably don't want to work for them, anyway.

"Do you have more than one resume, like a master one with all details and one page resume targeted to a particular position?"

Just a custom one pager specially made for each company. I show them the same level of special attention that I expect in return.

"In what order you present information in the resume: Objective, Experience, Skills, Education, Summary?"

1. Very short summary (with embedded skills) that pretty much says it all, "AJAX programmer, expert level in e-commerce, 100 projects completed, ready for next long term challenge in Big City, USA."

2. Applicable accomplishments in reverse chronological sequence. (Emphasis on "accomplishments".)

3. Degrees.

"Do you really think the resume layout matters more than the content itself?"

No.

"Which font do you use for your resume? Arial? Verdana? Webdings?"

Who cares.

"Do you prefer to maintain an online version of your resume?"

No. I'll contact them. I don't want anyone contacting me.

33. How to Write a Cover Letter

1. Write like you speak, as if told over coffee or beer.

2. Informal, but not too casual.

3. Right to the point; the first sentence is your summary.

4. No bullshit, you'll go straight to the garbage.

5. If it sounds like bullshit, it is.

6. Short. One minute good. Thirty seconds better.

7. Tightly targeted! It's about them as much as you.

8. Perfict speling and grammer.

9. Highlight what's important to them. (Do your homework.)

10. Enthusiastic without sounding phony.

11. Have friends read it. Get feedback.

12. Does it sound like a good quick description of you?

13. Have at least one differentiator. What makes you so special?

14. Strong finish with a call to action.

34. How good are most job applicants?

I was recently contacted by a head hunter for a job I thought was worth investigating. I wanted to talk to the company directly, but the head hunter made it clear that Step 1 was always a web-based programming aptitude test, no matter who you were. 20 questions. 18 correct was considered passing. They would only talk to candidates who got 19 or 20 correct.

So I took the test and got 20 correct (as I imagine many would as well). I thought it was very easy. The headhunter later told me that in 9 months, she had sent 52 people to the test, only 2 of us got 19, and I was the only one who got 20.

I'm not really sure what this means. That there are a lot of posers out there? That she wasn't very good at screening talent? That the companies seeking the best talent gladly pay $100 50 times to save their time?

I guess my biggest feeling is one of disappointment. It's just not that hard to become a good enough programmer to get 20 right every time. All it takes is study, passion, a lot of dedication, and a lot of hard work. I wish more people would do that. There aren't enough of us.

35. Taking an On-Line Aptitude Test

The entire test was about a fictitious language, I forget what they named it. Every question built upon the previous.

The first question was something like, "These are the definitions of variables: A string is... An integer is... A date is..." Then the question would be something like, "What can '15232' be? a. an integer b. a string c. a date d. any of the above e. a and c".

Another question might be "Here are the rules of precedence... What is (1 + 2) * (3 + 4) / 3 a. 4 b. 2 c. 7 d. 12 e. Can't tell for sure."

It got more and more complex, all in their own pseudocode. They covered branching, iteration, arithmetic operations, Boolean algebra, etc. Things most people here have seen a million times. Kinda like SATs for programmers.

36. I don't want to take a coding test

"As a programmer currently looking for work, I find this screening process to be draconian and insulting."

As a hiring manager, I look for someone who "gets things done". No surprise that this takes many things: knowledge, skill, smarts, creativity, experience, team work, and maybe most important: attitude. You have just self-identified your bad attitude. If you think this is "draconian and insulting", just wait until your knee deep in digital crap with customers barking at you all day long.

"I cannot vouch for it's efficacy as I've never used it when hiring people."

I can. Nothing works better. Period. If I have 30 to 60 minutes to find out how well you'll perform, I'll have you code and teach me what you've done. I don't care what you've done before, your state of mind, or even if your code ever runs or compiles. If you're any good at all, I'll find out. If you're a poser, I'll find that out, too. And if you don't want to play along, then either you're a poser avoiding exposure or a prima donna who is saving me a lot of suffering down the road.

I'm not trying to insult you, just give an honest answer to your (very good) question.

37. What will you do for a prospective employer?

Simple. Whatever it takes without being uncomfortable about it.

Things I will gladly do:

- phone interview with as many people as you like
- physical interview until I think you have enough information to decide
- share any of my work face to face
- on-line third party evaluation (code test, personality, etc.)
- review any of your system with you
- code for you in person
- sharing on-line information (including all hn posts)
- doing up to 8 hours of free sample work
- provide a referral if they are a better fit

Things I strongly push:

- in depth discussion about the employer's situation
- reverse interviewing employees with similar jobs

Things I will gladly do, but only "after" a job offer:

- provide references
- drug test
- credit check

Things I will never do:

- have a generic resume (every one is uniquely targeted)
- post any resume on-line
- pay a fee
- physical interview with a head-hunter
- provide anyone else's proprietary information
- "bad mouth" anyone else

38. Should I show my code to an interviewer?

Don't ever do this. Just a few reasons...

1. You say "during the application process". Have you even met the hiring manager at this point? Worse yet, have you met "anyone" or are you just stuck in some automated process? Never forget that employment application is a two way street; you're evaluating them as much as they're evaluating you. They haven't earned the right to see your code yet.

2. This approach should be a red flag to any applicant. They want to see old code to evaluate you when they are 64 better ways to do that? Something's fishy here - someone has no idea what they're doing.

3. Do not allow anyone to read your code out of context. You need to be there to explain the background, motivation, approach, and answer any questions. Cutting and pasting preempts yourself. Bring hard copy along only after the process has progressed far enough.

4. Only bad things can happen. This is like being the first the mention a number - you can't win. If someone wanted to hook up with you but asked for a dirty underwear sample first, would you give one? This is the same thing.

5. If your code is proprietary, who knows where it could end up? You may also be violating confidentiality agreements. Not worth it.

6. The employer-employee relationship is a never ending tension. You will always be "negotiating" money and working conditions, whether you realize it or not. Giving in to such an unreasonable request so early in your relationship marks you as a chump. You may never recover the equal footing you need (and deserve) in the ongoing relationship.

7. If they have a problem, move on. You're probably saving yourself a lot of trouble down the road.

You sound like a competent developer who should have no trouble finding a job without bending over. So don't.

[Aside: Because of this kind of thinking, I never give references before being hired. In essence, I'm telling the prospective employer, "You decide." They then have the right to rescind the offer if they don't like a reference.]

39. Why's it so hard to find good programmers?

"In fact, one thing I have noticed is that the people who I consider to be good software developers barely ever apply for jobs at all."

Good point. The best developers I ever hired were (a) already working, (b) not looking, (c) referred, and (d) without a current resume.

Therefore, the people who I consider to be good software developers probably don't have a current resume.

Therefore, the top 1% of good software developers probably don't have a current resume.

Therefore, if you have a pile of current resumes, it probably includes none of the top 1% of good software developers.

Therefore, if you're hiring from current resumes, your probably "not" hiring the top 1%.

[The only thing worse than sloppy probability and statistics is sloppy logic. But that's OK, because I'm not in the top 1% of either.]

40. Why do coding tests?

"No, coding on the whiteboard or on paper, or even the 5 minute exercise on the laptop is not really coding."

It doesn't matter whether or not it's "really coding". All that matters is how effective it is in evaluating your candidate.

I have interviewed over 2,500 devloper candidates and every single one has had to code with pencil and paper, in a room alone for 15 to 30 minutes. This has always been, by far, the most effective thing I could have done.

I never cared what they actually wrote. I never once found out if it would even compile or run. And I never cared. The only purpose of the coding problem was the "discussion afterward". This told me volumes.

Given a person, a discussion, a problem, and 1 to 3 pages of written code, I could ask many questions focused on a single issue and take it any direction I wanted. And learn what I needed to know about that person...

How did they attack the problem? What did they feel comfortable using? How did the deal with the situation? What kind of attitude did they have? How much did they enjoy dealing with the problem and discussing it? How well did they defend their choices? How willing were they to take criticism? How willing were they to stand up for what they believed in?

These are the things I want to know now. Not 3 months after they start working. Programming with pencil and paper and discussing afterward is the best way I've ever found to find these things out.

41. Why do a coding test on a white board?

"on a white board, no syntax errors, compilable"

Huh? You're worrying about whether or not what you write on a white board will compile? How do you know? Press a magic button to OCR what you've written, download it into some computer somewhere, and compile it? Sounds like you're interviewing at firms with technology I didn't know existed.

Your interviewers have you code on a white board so that they can evaluate "you", not your code. They want to see how you handle a problem, how you approach your work, how you think on your feet, how you deal with interaction, and how your intelligence and experience applies to their business. Anyone who worries about whether white board code actually compiles is missing the point. If they're more interested in perfect syntax than embracing you and your potential contribution, then you don't want to work for them anyway.

I think you're making this too hard on yourself. Memorize nothing. Just be yourself. Programming is like riding a bike; once it's part of your DNA, you don't have to worry about it. Just relax, trust your inner programmer, and let this become a self-solving problem; some jerks may reject you, but the right fit will come when someone sees who you really are.

42. Why do interviewers give code tests?

Here is the dirty little secret about evaluating the code you write in the interview:

What is on the piece of paper (or editor) doesn't make a whole lot of difference.

What happens when you and I talk about what you wrote tells me almost everything I need to know.

Less than one percent of interview code I've ever seen even compiled. What I was really looking for was:

- Did he understand the problem?

- How did he approach the problem?

- Does he appear as if he has any idea what he's doing?

- Can he explain what he has done?

- Can he defend what he has done?

- Does he understand the concepts of order, cleanness, iteration, branching, recursion, etc., etc., etc...

- Based on this small sample, do I think he can do the work we need done?

- Do I like him? Will he fit in and be a good team member?

So assuming they interview like I did (a big assumption), here's the good news...whether or not you'll do well has already been determined. So don't bother to "cram" between now and Tuesday. Relax. And have fun. Be yourself and it will all work out OK (one way or the other).

43. What is an example of an interview test?

Here's one of my favorite examples that can work for almost any language.

Remember, before any of this happens, I have already helped the candidate get comfortable and have made the purpose of the exercise clear: to assess where they're at and how/where they might fit in. It is not a test. Just an exercise to help us both. I offer them a soda or coffee, a little privacy, and this little problem...

You have an array. Call it "a" or whatever you want. It has a bunch of elements, numeric, alphanumeric, or whatever. You decide. Sort it. Without using a second array or a pre-existing function or routine. While I'm explaining this I'm sketching it out with my own pencil and paper. I suggest that they sketch out what they want to do themselves and then write some code (in the language being evaluated) or just pseudo code for general purposes. Be prepared to discuss whatever you want to present. Don't go nuts, just a few pages and 15 to 30 minutes. And have fun.

When I return, I have them explain how they approached it. (Here's what my code will do...) Then we go through the code line by line. At this point, it's incredibly easy for me to ask questions, such as...

```
Why did you name that variable that name?
Why did you use a for loop?
How else could you have done iteration?
How would you do it with 2 loops?
How would you do it with 1 loop?
Which variables are global?  Which are local?  Why?
Why did you reuse the variable "i" in the inner loop?  (Oops)
How can you make it faster?
How could you make it clearer?
How would you change it if you knew the probability of the original order?
How would you refactor this?
How would you extend this to do...?
Which code would you put in a library for reuse?
```

You kinda get the picture. No 2 interviews are the same. Imagine the programmers you already know having this discussion with you and how much you'd learn about them.

There are no right or wrong answers, just learning. Which is what you want.

This simple test eliminates the 90% of applicants who are not suited for this work and 100% of the posers. You can tell right away who they are.

On the other hand, good programmers shine on this. It's actually fun to hang out and talk about this stuff with them. I've even had candidates email me later with revised code based on our discussion. Those are the motivated ones. Big points for that.

44. How can I do better on interview tests?

"My biggest pet peeve with these types of tests is this: there are a lot of companies out there, and I'm sending out resumes to each one that I can find."

That's your first mistake.

Why should a company take you seriously if you don't take them seriously? Every resume you send out should be custom written and carefully crafted to address their requirements, not yours. Do you do any research into the companies you're applying to? You should.

Rather than sending out 50 stock resumes, you'd probably be better off picking the top half dozen or so opportunities and do everything you can to stand out. This includes a custom cover letter, follow up emails, thank you notes, etc., whatever it takes to make the connection.

"I simply do not have the time to write fifteen code samples a day, just because you want to evaluate me against your coding test. Period."

There's your second mistake. Misdirected attitude. Again, you should be focused on their needs, not yours. If you don't have time to do 15 poorly, then just do one or two very, very well. No employer or recruiter cares that you've chosen a shotgun approach.

Give of yourself without expectations of something in return. You may be surprised at the "unexpected" dividends you'll reap later.

For what it's worth: I have conducted over 2500 technical interviews and every single one of them has had to code, regardless of experience or background. What good is a programmer who doesn't want to code at the one point when it would provide the most benefit to everyone involved? In my experience, the best programmers were the most eager to give it a shot.

45. Should I send a Thank You note?

"...it is 100% necessary to send an email..."

I take it a step further. I always send a hand written Thank You Note by snail mail. Always. I have never failed to do this.

Why? Because if someone has spent an hour of their time (I don't care if it's on company time) to potentially change my life, the least I could do is take 5 minutes to share a sincere Thank You. I always take the time to hand write it, I always make it personal, and I am always sincere. (If you're going to be even 1% phony, don't bother, people will see right through it).

There are some nice byproducts. It demonstrates professional courtesy. It demonstrates good communication skills. It demonstrates good customer service. It gets you remembered. Every time.

But I don't do it for these reasons. I do it because it's the right thing to do. It is never "annoying" or a "detriment". Almost everyone I have ever sent a hand written note has told me that they really appreciated opening it and reading it.

Sometimes I even take it a step further. Several times, I have asked public speakers what their favorite restaurant is, and then sent a gift card in the Thank You note. I really want them to understand what a difference their contribution has made in at least one attendee's life. The gift card is a nice idea because I know they will use it and enjoy it, and maybe even think of me that night. I would never send a gift to an interviewer - that's not the same thing.

46. How can a company blow a job interview?

Top Ways for a Company to Blow a Job Interview

1. Make me wait in the waiting room past our appointment time. My time is as valuable as yours.

2. Make me meet with Human Resources first. My time is as valuable as yours.

3. Make me fill out forms. This can be done in advance.

4. Don't shake my hand. (I have no idea why this happens, but it does.)

5. Begin the conversation with anything other than, "Hello," or "Nice to meet you." Again, why would the first words of any interview be, "How would you handle mass emails?"

6. Don't give me your business card. As a job applicant, I'm just as important as any vendor or business associate. This is a good chance to demonstate it.

7. Criticize my job history. Here is the reason I've had 9 jobs in the past 5 years: Because there were poorly run businesses, assholes, and mismatches. The reason I'm here is to try to fix all that. Move along.

8. Expect me to have 10 years experience in every possible technology your entire enterprise uses. I will learn what I need. Promise.

9. Ask me how many gas stations are in the United States, how many ping pong balls fit in a bus, or why manhole covers are round. Frankly, I don't care. There, now you have the results for both your I.Q. test and your personality test.

10. Don't follow up. I know that I'm probably not the only person your interviewing and this may take some time.

Extend me the same courtesy you would the window washers. Don't make me email you every week.

47. Why doesn't anyone call me back?

1. There is no position. We're just "feeling out" the marketplace.

2. There is no position. This was a headhunter building his database.

3. We were planning to promote from within, but HR made us post the position anyway. We didn't read or respond to any of the resumes.

4. We already had the perfect candidate, but HR made us post the position anyway. We didn't read or respond to any of the resumes.

5. We posted the position as required by HR, but when an executive saw it on the intranet, he made us hire his son/nephew/family friend. We didn't read or respond to any of the resumes.

5. We were planning to hire someone, but by the time the resumes started arriving, the perfect candidate presented himself. We didn't read or respond to any of the resumes.

6. We were planning to hire someone, but the budget was cut. We didn't read or respond to any of the resumes.

7. We got 1,200 resumes in 2 days so HR ran them through a filter with almost no correlation to potential suitability for the job. Your resume didn't get through the filter. Next time, add buzzwords from the ad.

8. Your resume made it through the HR filter, but we only had time to read 20% of them. Yours wasn't pretty enough.

9. Your resume didn't stick out in a field of many that did stick out. You probably should have some kick-ass differentiator FRONT AND CENTER.

10. We read many great resumes. Yours was substandard compared to many of them for one or more of many

possible reasons. Have 5 friends proofread it and give you brutally honest feedback for next time.

11. Your resume sucked but you don't. Find 5 friends. See #10.

12. You interviewed well, but someone else absolutely kicked ass. We loved him/her. Tough break for you, I guess.

13. You didn't interview well, but we can't really put our finger on it and don't have time to respond. Tough break.

14. You interviewed well and are still under consideration. But we are waiting on corporate for 27 other things. You'll probably find another job before we get back to you.

15. You really do suck. (No you don't. Chances are the process never got this far.)

48. Should I go into management?

As a former developer turned manager turned developer again, a few suggestions:

1. "Never" lose your developer mindset. Personally, I have great difficulty respecting a non-technical manager of technicians. I bet I'm not the only one.

2. Manage things. Get good at project management. Very good. A good plan, agreed upon, well built and administered, will always be your friend. Something for everyone to fall back on when things get hairy.

3. Don't "manage" people, lead them. Preferably by example.

4. "Everyone" has problems. Now that you understand that, don't let people's problems interfere with their work or nothing will ever get done.

5. Learn the difference between issues and details. Focus on the issues. Don't waste too much time on the details. And get your team to do the same.

6. When all else fails, communicate. When all else goes well, still communicate. Never underestimate the power of communication. You'd be surprised how much slack others will cut you if you're simply open and honest with them.

7. Treat everyone else the way you'd like to be treated. (This goes for everyone, not just managers.)

8. Listen at least as much as you talk. (This also goes for everyone, not just managers.)

9. Get stuff done. And have fun doing it.

10. Lighten up. You'll be fine.

Chapter 4

Work Habits

49. How do you achieve laser focus?

The single most important thing I do to "achieve laser focus and concentration" is to work in such a way that I don't need "laser focus and concentration" to get my work done.

This has to be done the night before.

I always quit all online work at least 2 hours before bedtime and print whatever I'm working on.

Then I go into any other room with program listings, blank paper, and pens (especially red!) and plan out all of tomorrow's work.

All analysis, design, and refactoring must be done at this time. I do not allow myself to sleep until the next day's work is laid out. I also do not allow myself to get back onto the computer. The idea is to have a clear "vision" of what I am going to accomplish the next day. The clearer the better.

This does 2 things. First, I think about it all night (maybe even dream about it). Second, I can't wait to get started the next day.

I always wake up and start programming immediately. Once I get going, it's easy to keep going. Any difficulties are probably because I didn't plan well enough the night before.

50. My Working Guidelines

1. Start with the answer, then work back.
2. Name your variables so that anyone will know what they are.
3. Name your functions so that anyone will know what they do.
4. Never write the same line of code twice. Use functions.
5. Assume the user doesn't know what they want.
6. Even if the user knows what they want, assume they can't verbalize it.
7. The user always knows what they don't like. Prototype often.
8. Be prepared to dig down as many levels of detail as needed to understand.
9. When you're stuck, turn off your computer.
10. Don't turn your computer on until you have a specific task.
11. Beauty is important, but delivery is more important.
12. No variable should be fully contained within another variable.
13. All variables should be at least 3 characters long.
14. Use the right tool for the right job.
15. Almost any tool can do the job. Some are better than others.
16. Benchmark often in order to learn what happens under the hood.
17. Try something that's never been done. It may be easier than you thought.
18. Remember the patterns you've used before. You'll use them again.
19. Keep it extremely simple at first. Complexify as you go.
20. Code every day.

51. How to you start a new project?

How I Write Code:

1. Copy the smallest piece of code I've already written and change it slightly to do something really small. Get it working perfectly.

2. Add a little bit. Get it working perfectly.

3. Add a little bit more. Get it working perfectly.

4. Repeat until it's not so easy to add anything, even a little bit. Print a hard copy. Holy crap! Did I write all that?

5. Mark it up like crazy with a red pen. Combine similar code into common functions. Rename variables. Rename them again. Rename them again. Rename them to what they originally were, but without vowels. Restructure unwieldy functions. Rewrite the stuff I can't believe I actually wrote myself.

6. Sit down at terminal and enter changes. Save every version (even though I never look at it again). Get it to work perfectly (again). Holy crap! What did I do? It's totally broken! Debug, debug, debug. Good. Now it runs perfectly. Time to add more...go back to Step 1. (Except every 5th time through this loop: Scrap it all and rewrite it the way I should have in the first place.)

7. Repeat until dead.

52. Why do you use such simple tools?

This question reminds me of the greatest cook ever, my grandmother. She used no technology whatsoever. All of her tools had been her mother's which were probably manufactured in the 1800s. She chopped everything by hand in a wooden bowl. If anyone else helped her with the chopping, everyone at dinner could tell. She never used pencil or paper and measured nothing. She stood in line at the farmer's market, the butcher, or the grocery store and inspected every item. And absolutely nothing I have ever eaten since, in any restaurant or home, has been remotely close to hers. It was magnificent! And I miss it so much.

I'd like to think I have almost as much passion about my work. I use the most primitive tools, 24 x 80 green screen editor, no framework, no IDE, no debugger, and mostly pencil and paper. I savor every byte just as I imagine my grandmother savored every little detail of her cooking. I'm not trying to save time or be fast, I just aspire to creating Grandma-quality software. I only hope my software brings someone the same joy her food brought all of us.

I've used many different tools. And I rarely care how fancy they are. Ironically, the simpler, the more joy I have found along the way.

53. How fast do you work?

Very fast. Let me explain.

I work in 2 modes: (A) At the computer and (B) Away from the computer.

When I'm in Mode A at the computer, I'm cranking out lines of code, testing, revising, testing, revising, etc. This process must be very fast. Several hundred lines of code (or whatever) in less than an hour. A complete cycle in less than a couple of hours. My guideline is that if I'm not working that fast, then I must not be prepared to work that fast, so I don't deserve to be at the computer. I should be in mode (B).

Mode B is generally much slower. Reviewing code, specs, or notes. Refactoring code. Laying things out with pen and paper. When I have enough work clearly laid out, I know it's time to get back to the computer and return to Mode A.

The most important thing for me in Mode A is to see results, any results, quickly and often. It doesn't matter how correct anything is, just as long as it's progress (or sometimes, reverse progress). I like to think of programming as making incremental progress in micro jumps, evaluate where I'm at, and go for the next micro jump.

Some of the best advice I ever got was from a prolific artist friend of mine who claimed, "I paint every day." So I started coding every day. But that wasn't enough. Now I make progress every day.

There are many definitions of progress. Sometimes I copy a few hundred lines of code, make a few changes, spit out a new app, and then start applying micro changes. Other times I decide that I need to see today and find a way to get there. Things don't always work out as planned, but that's OK. As long as tomorrow's starting point is beyond today's, I'm satisfied.

That's my definition of fast. Not sure that was what you were asking, but I hope that paints you an accurate picture.

54. What does your IDE look like?

55. How do you work?

BIG disclaimer: I have NO formal training.

1. Tools. I generally shy away from tools. I just don't like using anything that makes me more productive when I'm programming. I prefer to type out every line of code, occasionally cutting and pasting from the same program or something similar from the past, but not very often. I want the writing of code to be painstakingly slow and deliberate. Why? Because productivity is not the objective. Becoming one with my project is. I may not start as fast as others, but that doesn't matter. It's not how fast you start, it's how soon you deliver a quality product. Like memorizing a speech, cranking out the code by hand makes it firmware in my brain. It's not unusual for me to crank out 300 lines of code and then be able to reenter them on another machine from memory. So when it comes time to debug, refactor, enhance, or rework, those cycles go very quickly; that code is already in my brain's RAM and it got there the hard way.

2. Simple Algorithms. Yes! I love this example:

```
* EnterOrders 08/31/10 edw519
*
return();
```

I now have a working programming! You say you want more features? No problem. Let's start enhancing it.

3. Debugging. I don't. I've seen 25 different debuggers and I hate them all. Print() is my friend, especially beyond the right fold. Better yet, write code that doesn't need to be debugged (See #1 & #2 above.) (Of course, this is much harder with "someone else's" code.)

4. References. Don't need no stinkin' manual. I have become extremely adept at about 4% of what's available to me,

but that 4% accomplishes 98% of what I want to do. (OK, I'll refer to a manual when I need something from the other 96%, but that doesn't happen too often.)

We could talk about all kinds of other things too, like variable naming, how to iterate and branch, and never typing the same line of code twice, but this was a nice starting subset.

56. How do you stay so jazzed?

I've been doing this for 32 years and I'm more jazzed than ever.

Wanna know the biggest difference between you and me? I'm pulled. You're pushed. Let me explain...

I love building stuff. Nothing gives me a bigger rush than getting something working the first time . But I couldn't care less about the technology. If an abacus, two tin cans on a string, or some BASIC code on a Kaypro II did the job, then that's what I'd use.

What I really care about is how my software is used. And who uses it. There's an endless stream of people who need stuff and an endless stream of problems to solve. For individuals, groups, and businesses. When I encounter a new problem to work on, I use whatever I can apply in my tool box. Sure, I have to upgrade that tool box every so often because I need more to solve my problems, not because I love the toys so much.

You sound like the opposite. You love the toys and look for places to use them.

My suggestion: Take a break from the technology and put yourself in more situations where people can share their problems. This will give real human meaning to the technology. I bet you'll be chomping at the bit to build something for someone in no time. For me, being pulled by a demand motivates much better than being pushed by a supply. Maybe it can be for you too.

57. How do you make better use of your time?

"Work effective hours, not necessarily long hours..."

The best trick I have ever figured out for doing this is to separate all activities into "in front on my computer" and "away from my computer". If you are working ineffectively in either mode, switch modes. If you are still working ineffectively, consider a break.

I often sit in front of my computer, writing code, refactoring, or testing and realize that I'm getting nowhere. Then I conclude that the reason I'm not making progress is because I'm not quite sure "what" to do.

Determining what to do is an activity that I have found much more effective away from the computer. So I log off, grab pencil and paper and go somewhere else, anywhere else. As soon as I have something I'm sure I want to code, then (and only then) do I return to the computer.

This works both ways. If I'm away from my computer, but feel I'm not creative enough, then I just decide to write something, anything, and go write it. Sometimes just getting the smallest things done opens the doors to getting bigger things done.

58. How do you split your time up?

I'm guessing my split would be more like:

```
sales & marketing - 10%
analysis          - 15%
design            -  5%
development       - 50%
implementation    - 10%
support           - 10%
```

Testing is not a phase, it's included in everything. Implementation includes deployment, training, and documentation.

Sales and marketing were never problems. There is business "everywhere". I mention what I do everywhere I go (who doesn't), and find business almost anywhere. Everyone uses software these days and everyone needs "something". I've gotten 5 figure deals at parties, family functions, networking events, and mostly from word of mouth.

I love to hack, but I also love to talk about what I do. I may not be typical, but isn't helping people with cool tech what it's all about?

In my mind, the problem isn't the time needed for selling and marketing. The real problem is that this kind of business (consultingware, one-off packages, lite package with customization, whatever you want to call it) isn't scalable.

If my competitive advantage is better value because "the boss" (not some lightweight on an 800 number) is supporting you, then it's also my biggest limitation. I only have so many hours in a year. Sure, I can extend myself with advanced methods and technology, but sooner or later, I hit my limit. That's why it's so important to find a way to convert your consultingware business into a product business. Not an easy feat, but surely worth the effort.

59. How do you get unstuck?

Some of the things that have worked for me:

- Decouple analysis from coding. There are some things that should NOT be done in front of a terminal. Get a pencil and paper and go somewhere else. By the time you're done, you'll have plenty of stuff to code. (I have found that the main reason I get stuck is because I have not spent the requisite time "away" from the terminal laying things out. I'm always in too much of a hurry to "get back to work" before I'm ready.)

- Get a customer. They'll give you something specific to work on. If it's maintenance, all the more reason to get to just dig in and get to work.

- Ask yourself the question, "How am I making this too hard?" If you listen to yourself long enough, you'll probably get a good answer and a fresh approach.

- Reduce scope. Remove outlying cases. Solve only for the most probable case. Get that working perfectly. The process of doing this will probably shed a lot of light on how to set up structure that will also handle the outlyers.

- Back up your current version and the go wild on it with some crazy approach. Knowing you have a good backup frees you up all the more. At the end of the day, if you have something cool, keep it. If not, just restore your backup and throw away today's work. You may have wasted the code, but you didn't waste your time. You probably got the juices flowing again.

- Backup your current version and forget about it. Wipe the slate clean and start over completely. You won't have to worry about satisfying all the overhead you've already created. After a day or two, keep either the original version, the new version, or more likely, you'll have a new project: combining the best of both. In any case, you'll be busy working again.

- Set the project aside and work on something easy and fun that no one needs and provides little value to anyone.

The byproduct is that suddenly, you'll discover you're working and enjoying it. The next thing you know, you'll "want" to go back to the more difficult project.

- If you're stuck on something, post your dilemma on-line. Read the responses. You may get a different approach that you can play with. And even if you don't, you won't feel so alone. That may help.

60. What's most important about work?

I absolutely "love" what I do and can't imagine doing anything else.

"However"...I have had very few jobs where I was happy. I think there are 2 primary reasons.

1. I want to work on what I want to work on. When I work on anything else, all I can think about is what I really want to work on. In a job situation, I rarely work on what I want. (OTOH, a job that has me working on what I want is usually a great job.)

2. I want to work when I want to work. Sometimes, 8 to 5 works, often it doesn't. We were simply not meant to sit in cubicles without windows all day long. Enough said.

There are lots of other things most of us don't like (difficult people, crappy code to maintain, poor management, difficult deadlines, etc.), but those are all part of the territory. I could live with them if I could work on what I want when I want.

61. How do you get things done?

I like to keep it simple. My list has 1 item on it. I work on that until either it's done (often) or I struggle so much with it that I decide to change plans (rarely).

For the last 2 days, I've been writing a model configurator that explodes input parameters into individual objects. I probably have 8 or 9 things dependent on this (not really sure yet), so I plug away until done. Then I'll figure out the new only thing on my list.

I've tried every conceivable "productivity hack" and nothing has worked as well as this. I have scratch pads, paper on the wall, 20 colors of markers, and all kinds of automated tools for scheduling and planning. I've varied my diet, my exercise routine, my daily routine, and almost anything else I could vary, and none of it really mattered. All it ever really did was take focus away from the real task at hand.

Just identify your critical path, remove it, and repeat forever.

I started with this and fine tuned what worked for me:

http://paulgraham.com/procrastination.html

Other inspiriation:

"I see only one move ahead, but it is always the correct one." chess master Jose R. Capablanca

62. How do you keep track of your thoughts?

It's really simple for me...

1. I write everything down in an unlined spiral notebook with perforated detachable pages.

2. I file every page into a labeled green file folder in a file cabinet.

3. I keep all of it.

I've been doing this for 30 years. I have "everything" I ever wrote. If fills 3 two drawer file cabinets.

I don't print and save anything which is already stored digitally. I hardly save much else.

About once a month I pull out a folder a go through it. Obviously, there's a lot of stuff that appears to be of little use now, but I never fail to find "something" of value.

I give away or donate any that is replaceable (which includes all books). But not my own writings. I don't remember how I handled that issue 12 years ago, but I do know that I can find all my notes on it pretty quickly. This way I don't have to remember every detail, but I always have my younger self and much of my experience as a resource at my disposal.

63. How do you boost your creativity?

The single most important thing I do to "boost creativity and/or productivity" is to work in such a way that I don't need to "boost creativity and/or productivity" to get my work done. This has to be done the night before.

I always quit all online work at least 2 hours before bedtime and print whatever I'm working on.

Then I go into any other room with program listings, blank paper, and pens (especially red!) and plan out all of tomorrow's work.

All analysis, design, and refactoring must be done at this time. I do not allow myself to sleep until the next day's work is laid out. I also do not allow myself to get back onto the computer. The idea is to have a clear "vision" of what I am going to accomplish the next day. The clearer the better.

This does 2 things. First, I think about it all night (maybe even dream about it). Second, I can't wait to get started the next day.

I always wake up and start programming immediately. Once I get going, it's easy to keep going. Any difficulties are probably because I didn't plan well enough the night before.

With proper planning, "getting into the zone" becomes a much smaller problem.

64. How do you stay productive?

1. Do not have access to the internet on your work machine. If you don't have 2 computers, get a netbook for < $300 and connect it to the internet. They should be in 2 different workstations, ideally in 2 different rooms. The thinking is that if you have to get up, you'll only do it if it's really necessary. It works pretty well.

2. You should have 2 modes: coding and not coding. For coding, you should be at your desk coding. For not coding you can be anywhere, but "not at your desk". One of my biggest problems is that I often find myself in one mode when I should be in the other. If you're having trouble writing code, then you probably don't know what to write. Grab source code listings, pen, & paper and "get away from your computer". Don't come back until you know exactly what you're going to be working on. Better yet, until you're "dying" to work on it. On the other hand, if your doing analysis and are stuck, stop. Go back to the computer and code something, anything small, just to get going.

3. End every day in analysis mode. Don't go to sleep until you have tomorrow's plan ready. You should wake up knowing exactly what you're going to be working on and excited to do it.

4. Never text or IM when working. Have the cell phone nearby only for emergencies. For email, go to the other computer once an hour (see #1 above).

5. Try 48 minutes on, 12 minutes off. For long coding sessions, this works pretty well for me:

http://successbeginstoday.org/wordpress/2006/09/the-power-of...

6. Ipod.

65. How do you combat work overload?

My single biggest secret for continuing to get things done, often working 12 hour days and 6 day weeks, is to love what I do.

I do not think of it as "work", as something to "get through", or as something difficult, unusual, "overload", or temporary. Quite simply, I sit at my computer 12 hours per day, every day, because "I want to", I love what I do, and I can't imagine doing anything else. I have been working like this for years.

In fact, I feel sorry for anyone else who doesn't feel this way. What a sad life to be spending so much time doing something that you have to force yourself to do.

Some of the things I have arranged in my life to enable me to do what I love:

I eat well. It's a great idea no matter what your circumstances are.

I also exercise regularly, 20 minutes, 3 to 6 times per week. I mix it up and only do things I love: 5 rites, pushups & pullups, jogging, swimming, heavyhands, shovelglove, bodyweight exercises, and even a day or 2 at the gym on their machines.

Breakfast at my desk, lunch away from my desk, dinner and Jeopardy with my SO every night without exception.

At least one date night per week, with an extra beer or two just in case it's a chick flick.

I never text, tweet, blog, IM, or facebook. I do respond to voicemail and email regularly. Everyone I know understands this.

Ipod, radio, and 3 cats keep me company.

I like to work in 48 minute bursts, with a 12 minute break each hour for email, internet, snack, or anything else.

66. Why work more hours?

"In addition to the usual work-day schedule, I expect all of the members of the group to work evenings and weekends. You will find that this is the norm here at Caltech."

Then you're doing it wrong at Caltech.

We are often quick to assume that MoreHoursWorked = MoreWorkGettingDone. This is true up to a point, but false beyond that point. Personally, I believe that evenings and weekends are usually beyond that point.

I used to work 90 hours per week. But when I decided that I needed to get more done, I started working 60 hours per week. Results per hour and quality of results have both improved dramatically, so I'll never go back. And I would never work for anyone who doesn't understand this.

67. Why be fearless?

This reminds me of my first partner in our software/consulting service business. He was absolutely fearless.

We would always arrive at appointments very early so he had an excuse to "poke around". He'd ask anybody, the receptionist, someone in the breakroom, even the janitor. He'd see what was going on in the parking lot, the loading dock, even in the warehouse or factory. Seeing him in a business for the first time was like watching a kid in a candy store.

In our first meeting, he always knew something about the client's business that they didn't. He'd say things like, "Automating the inventory won't help if Fred and Jean are counting 2 different things." This always led to interesting discussion and often, follow-up business.

Once he even spent a week of his own time on third shift, going over procedures and reports with factory supervisors. They didn't know who he was; they just figured someone from the main office sent him. He did a complete analysis in Excel which we used in a proposal. That got us hundreds of thousands of dollars worth of work.

I often challenged him, "You can't just do that," I would say. To which he would respond, "These people need help and don't even realize it. We have to find a way to show them." Then the inevitable, "It's better to beg for forgiveness than ask for permission."

Looking back, it didn't always work. It pissed off some people and burnt those bridges. But when it did work, we often concluded that nothing else would have.

I learned a lot in those days. I'm still not as fearless as my partner was, but I'd like to think I'm getting there. Thanks for the memories.

68. Can programming be boring?

Maybe "boring" is not the best word. Maybe we are really talking about "more fun" vs. "less fun".

Naturally, some things are more fun that others, but I am never bored in my startup.

Frustrated sometimes, yes. This week I lost a whole day because I had overlooked something simple one day last week. Had to retool the whole stupid thing when I really wanted to build the next level up. So the exciting part had to wait a day. No big deal. It happens. But was I ever "bored"? Hardly.

A little background. I sat in class bored to tears for 17 years. Then, I did work in 86 other companies (none of them mine) before I started this one. I have a clear vision of what I want and a fairly clear idea of how things should work. I love both the technical details and the people part. Usually, I can't wait to get to the next thing.

The only thing I really don't like is when a client calls for me to fix something on their crappy system. Shifting gears sucks.

Boring? No, I just don't see it.

69. How do you manage your time spent?

My solution? I simply stopped worring about how much time I spent doing or not doing something.

I started focusing on one thing only: the delta between what I planned to have complete at the end of each day vs. what I actually completed that day.

On Thursday, my plan was to have items A, B, and C complete before I knocked off. A & B were done by noon. I overlooked 2 prerequisites for C and had to go back and do them, then do C, which took twice as long as I expected. I didn't finish until 2 a.m. I also spent x hours on line. So what?

Sometimes going online gives me a break. Sometimes it gets the juices flowing again. And of course, sometimes it's just a waste of time. But it doesn't matter.

What's better, spending 6 hours online and getting everything done or spending 2 hours online and not finishing?

We often forget that (time spent) != (work accomplished).

Stop worrying about how much time you spend planting seeds and focus more on finishing each day's harvest.

70. How do you capture good ideas?

You can have an interesting idea "any" time. Problem is you may not even realize that you just had an interesting idea when you had it.

The journal is an excellent idea. I write everything down. When I look it over later, I'm usually embarrassed that I could have thought of anything so lame. But every once in a while, there are a few gems in there.

I always have pencil and a small notebook next to the bed. The best time for me to get good ideas is as soon as I wake up. The second best time is right before I go to sleep. I carry index cards and a pen at all times during the day, just in case.

The other important ingredient in good ideas is to get out there and experience things. All the time. You never know which inputs will spawn ideas, so get lots of inputs.

My best ideas have usually come when I see how something is and think, "There must be a better way." Then I let it lie dormant inside and trust my "inner self" to come up with something when I least expect it.

(My best hacking ideas ever came from the first time I saw a code generator. Simultaneously I thought, "That is so cool!" and "It can be so much better than that.")

71. How do you get good at programming?

I believe that there are two ways to get good at anything, "push" and "pull".

Push: You learn from books, classes, mentors, and studying examples, then apply what you have learned.

Pull: You have a problem that you must solve, then you learn what you need, any way you can, to build the solution.

I suppose there are pros and cons of each method, and I imagine that many people here have used some of both.

For the record, I am 100% Pull. I have absolutely no formal training. It took me 2 years to find my first job and then I was thrown into the deep end. It was simultaneously frustrating and exhilarating. There were so many times I didn't know what to do or didn't have enough "tools" in my box. So I had to figure it out and find sources of learning. But I always did. Any when I got that first thing working and then saw my customer's eyes light up, I was hooked.

Your CS degree may make you think that you're a "push" learner, but may I suggest that you adopt a "pull" approach. Forget what you think you know and find a job or a project or someone who has a real need. Then build what you need. You a several advantages over me: (a) It shouldn't take you long to find that job/demand/customer. Just keep looking. (b) You already have tools in your tool box, maybe not the right ones for the job, but you have "something". And (c) It's easier than ever to adopt a "pull" approach. Help is everywhere.

You may feel frustrated, but I don't think you have a problem at all. You're in a great (and very normal) situation. Just adjust you attitude, find something to build, and do it.

72. How do I rise out of the ordinary?

I have been in your situation many times and have felt the same way; welcome to the rest of the world.

Face it, those of us who are uncomfortable with the status quo and want more than the masses are outliers. This is a good thing! Here's what has worked for me...

Rising out of the ordinary and getting ahead is all about "demand", not "supply".

Supply: You can read all the blogs you want, read technical books, and study new languages and so what? You are supplying yourself with things that may or may not make any difference and no one else cares.

Demand: Find out what needs are not being met at your company. You can easily do this all the time without even being noticed, "under the radar" as they say. (You don't have to go to your boss looking for something.) They are desperate needs not being met everywhere; all you have to do is look. It may be a user, a customer, a way of doing business. Then fill that need. It's that simple. You will force yourself to learn whatever you have to to get that job done. Then do it again. And again. Before you know it, people will automatically know to come to you when nothing else works.

Business is about getting things done. Learn how to do that whether it's officially sanctioned or not. You will quickly rise above the masses and your life will never be the same.

A few notes:

1. Most people won't even realize what you're doing. For those that do, most will approve. For those that don't approve, ignore them.

2. With this approach, you will optimize your learning. You will learn what is needed, not what you think is cool.

3. You will become much more valuable for two reasons: you'll expand your skill set and you will convert yourself from a doer to an achiever. The difference is subtle but huge.

73. You're Not the Problem, the Work Is

"Despite all this every time I sit down to code my brain turns to mush, somehow, and nothing gets done. I can always give my managers an intelligent explanation of why progress is so slow..."

Oh, so this happens at work...This exact same thing has happened to me many times and here is what I figured out...

The problem is NOT with you, it's with the work.

Like you I have scored excellently on interviews and tests and have written code so cool I even surprised myself. So we have to believe that we are as good as we think we are. That belief is what enables us to take on the hard projects. That belief is also what turns our brains to mush when our work is too many levels below our capabilities.

Sure, we all do work below our level every once in a while; we have to, that's the nature of the beast. But when you're doing low level work for too long (like at many jobs), you lose energy from the lack of stimulation. We then mistake that lack of energy for many other things: our skill, our commitment, even our health.

Here's what I've done in situations like yours: build something cool on your own time. It will get your juices flowing and you'll be your old self in no time. Of course, you'll want to work on your own project instead of your employer's, but that's another problem.

74. What are the basic modules in your library

1. basic form
2. form with multi-valued lines
3. form with data set (multi-valued parent/children)
4. form with grid
5. tabbed form
6. form with skins
7. batch (loop thru something)
8. batch update selected records in a table
9. batch dump table records --> .txt or .xml file
10. batch .txt or .xml file --> update table records
11. batch file --> create & populate database table
12. cut an email
13. traverse internal tree function
14. traverse file index(es)
15. build html from parameters
16. build javascript from parameters
17. build .pdf from parameters
18. build hp esc sequences from parameters
19. benchmark a process
20. how does that syntax work?
21. which way is faster?
22. batch string parser
23. batch source code search
24. batch source code changer

I have more, but I don't have time to find them right now. OK, I think I'll add:

25. batch parse source code, identify routine for reuse

75. Why are you a "caveman" programmer?

Tell you why I don't like syntax highlighting (or any crutch). Ever since I read the chapter about Woz in Founders at Work. The thing that he thought made him so successful designing the Apple II: he knew every single little part of it intimately, like the back of his hand. That struck me like a lightning bolt. So that's how I feel about my code now.

If I want to do something, I read and reread and reread it over and over and over until I practically memorize it. THEN it's part of MY firmware, and that's when I really get insightful and productive. IDE's, syntax highlighting, etc. are just crutches that keep my real "hacking muscles" from developing. I won't use them. Black and green; that's all I know.

Similarly: I used to know every phone number I ever needed by heart until speed dial came out. One day last year, I had the very frightening experience of not being able to call a regular number from someone else's phone. I have never used speed dial since. Anything that impedes my "brain exercise" is something I do without.

76. What tools do you use for analysis?

Pen and paper.

This is a hard and fast rule that I have always followed and is absolutely critical to my success. Just a few of the reasons:

1. I firmly believe that analysis, design, and planning is much more effective if it is done "away" from the computer. These are totally different activities from programming and they take a different mindset and environment. That's pretty tough to do if you use computerized tools to organize.

2. I like to spread out my notes, plans, diagrams, lists, etc. on a table to work on them. I also tack them up on the wall above my work space, both at the computer and in the other room. I want to give my mind every opportunity to see the "big picture" when it's appropriate. Again, tough to do with a computer unless you have 5 monitors.

3. I carry my notes with me whereever I go. You never know when inspriration will hit, and I don't want to carry a laptop everywhere and wait for it to boot up.

4. Bedtime is critical thinking time (both at night and in the morning). I always have my notes and multiple colored pens with me in bed. Some of my best ideas have come at this time. I can't imagine the same thing happening with a laptop.

77. Every Task is Complete or Not Complete

This reminds me of my hero of project management, Tom DeMarco.

http://en.wikipedia.org/wiki/Tom_DeMarco

His approach was that every task is either 100% complete or 0% complete.

No task is ever "80%" complete. If you really think that it is, then break it down into smaller tasks, 80% of which are 100% complete and 20% of which are 0% complete. This way it becomes much clearer what is complete and what remains to be done.

I have never allowed anyone to report anything as n% complete for values other than 0 or 100. As soon as someone does, that's a pretty good sign that they may be confusing motion with action.

Remember an old discussion with Jim...

```
Me: Jim, how are we doing with getting Ansys ported?
Jim: Great, I have a bunch of calls into them.
Me: How are we doing on the Nastran port?
Jim: Wonderful, they said they'll get back to me next month.
Me: How about Dyna 3D?
Jim: It's going great, we're on their list.
```

Now imagine how much different that conversation would be if each question started with, "Which tasks are complete and which tasks are incomplete?"

78. Why do you love email?

I love email. Why? Because I'm a programmer and I don't want to be interrupted. I like to keep the day-to-day things simple because my work is anything but simple.

I use gmail and everyone I know has my gmail address. It has excellent spam filters and it's easy to manage. Best of all, it never bothers me. When I'm ready for a break, I check and manage my email. It takes 5 minutes, just enough time to eat an apple.

Family, close friends, and some business associates have my cell phone #. They know not to text me because if I'm working, I will not respond. If I'm not working, the only response they'll get is "ok". If the phone rings, I know it must be important to talk. Usually not for long, then back to work.

I do not IM. I do not Twitter. I can't imagine worse ways to ruin productivity. I IM'd for 3 days. I got nothing done, so I emailed everyone to never IM me again. Email me and I'll respond when I'm available.

I surf my favorite sites during breaks. When the break is over, on go the headphones and up comes my text editor. See you in an hour or two. Not before then.

79. Where do you like to sit in your office?

I prefer to sit with my users. As you can imagine, this concept is met with a bit of resistance in corporate America.

I want to dwell with them and be a part of their lives. I want to hear them complain about their apps, their customers and vendors, their bosses, and each other. I want to know what they go through all day every day.

When I sit and suffer with them, the resulting software is "always" better. All the meetings, prototypes, demos, specs, etc., etc., etc. have never been able to deliver the same knowledge needed to develop their apps.

On the other hand, I "don't" want to sit with other programmers, unless we're working on the same thing at the same time. I don't care about your problems, I have my own.

80. How important is office space?

Some people advocate: 1) private offices for each programmer, 2) free, nicely catered lunch every day (everyone eats together and bonds), 3) usual internet stuff - comfortable office, dogs, flexible schedules etc.

On one hand, a little voice inside of me cries, "Yes! This is what it takes to produce great software: a programmer-appreciative environment."

On the other hand, another little voice claims, "This stuff is all well and good, but nothing more. It's cosmetic, not functional, like putting perfume on a pig."

I have been in many situations with Class A office space, private offices, catered team-building breakfasts and lunches, etc., etc., etc. and was ready to jump out the window. Why? Because the work sucked and all the window dressing in the world wasn't going to change that.

Then I think back to some of my most favorite projects and remember the great work that we did. Sometimes in squalor-like conditions, sharing cubicles or tables, sitting in the server room with too much air conditioning or in the warehouse without enough, eating vending machine garbage and drinking old coffee because that was all there was.

Bottom line? The "work itself" is 100 times more important than the working conditions. Sure, everything being equal, I'd rather have nicer digs. But everything not being equal, I'll choose the better project every time, no matter what the environment is like.

81. How do you pick the best language for you?

I guess this is why I never post in language threads, we can go on and on all day. No one is right or wrong. Anyway, I'll try to address your comments.

There are so many different versions of BASIC; it's possible we are visualizing very different versions. I am accustomed to using INFOBASIC dialects as in JBase or IBM'S U2 line. Everything numeric is represented as strings (integers) including dates, times, and decimals, so there is no "typing".

As for efficiency, I used to benchmark like crazy, looking for the slickest algorithms and the best implementations of compilers and I/O routines. Not any more. Hardware has gotten so fast that I am more concerned with human time, not machine time. (Naturally, anything that runs 8 million times per second BETTER be fast.)

These days, I generate most of my code, so the verbosity of COBOL (if I used it) shouldn't be an issue. But it is. I have to drop down to the source so often and I work on the code of others often enough, that I am most concerned with getting the whole job done with the LEAST code. This is where "juniors" fall down the fastest (IMO); there's too many places for things to go wrong.

I know what you're thinking, if this is the case, why not learn even more advanced languages and frameworks with even less code. I dunno.

I guess I've "settled into" what works best for me (as everyone should). Is it the best? Probably not. Am I open to better ways? Probably. But not today. Too busy.

82. What are the advantages of working at home

Advantages of working at home:

1. Equipment I pick.
2. Furniture I pick.
3. Temperature I pick.
4. Lighting I pick.
5. Music I pick (without headphones!).
6. Clothing I pick (shorts in summer, sweatsuit in winter)
7. Food & drink I pick. (It's really good.)
8. Commute time = 1 minute per day.
9. No gas/car expense.
10. Almost no interruptions.
11. 6 people have my IP phone#.
12. 12 people have my cell phone#.
13. Everyone else --> email.
14. I check email when I'm ready, not them.
15. I set my task list (but still deliver as promised).
16. Can easily run errands any time.
17. Can easily do household tasks any time.
18. Much easier to schedule exercise.
19. See SO much more often.
20. 4 legged creatures make much better office mates.
21. People respect my time much more when I do visit the office.
22. Can more easily switch tasks.
23. Much easier to focus all the time.
24. I get twice as much done.

83. Any other advantages to working at home?

- I impose discipline upon others. Since you can't just poke your head in with an "idea", you have to think about it first (imagine that). If your email doesn't include enough data for me, I say so and hit "reply". Same for phone calls and voice mails. Signal to noise ratio increases dramatically.

- My performance metric is WorkCompleted / WorkPlanned because that's all people can witness. Not 101 other meaningless metrics like number of sessions running, time spent on internet, time spent BSing, time spent on cell phone, time spent at lunch, shirt color, or droplets of sweat on forehead.

- Wake up at xx:34 a.m. Start work at xx:37 a.m. (if I want to).

- Wake up in the middle of the night with an idea. Log in and work on it immediately.

- No meaningless team building, HR, or mandatory meetings.

- I eat what I want, when I want, where I want. No more vending machine or carryout crap.

- Headphones or speakers? Oh wait, never mind.

- Casual dress, gym shorts, or warmups. Again, never mind.

- Cats often make better companions than cubicle mates.

- I hate shaving.

84. How do manage to enjoy working at home?

The most important is the dedicated space with a door that closes. So you still "go to work", just with a very short commute. A few other things that I have found helpful:

- When you're in your office, you're at work, working.
- When you're not in your office, you're at home, not working.
- Work in 48 minute bursts, then take a break.
- Only check email & voice mail during your 12 minutes off.
- Put internet and work on 2 different machines (if possible).
- I prefer radio over pre-recorded music; it reminds me I'm not alone.
- Eat lunch out with a friend several times per week.
- Eat home meals away from your office.
- If you're local, go in to the office once per week.
- If you're not local, go to the office several days per month.
- Dinner with SO every night, no matter what you're working on.
- A regular schedule of sleep, meals, exercise, and work makes things easier.

85. How do you split your programming time?

My experience:

mode	% of time working on my own stuff	% of time cleaning up others' crap	% of time wasted on other B.S.
a. 9 to 5	10	40	50
b. consulting	30	40	30
c. start-up	100	0	0

86. Working at the Library

Maybe we're lucky in Pittsburgh, but we have the best of both worlds at the main branch of the Carnegie Library. You can take a tour of where I work 2 or 3 days per week:

http://www.clpgh.org/locations/main/tours/virtualtour/

It's fantastic. It was built by Andrew Carnegie in 1895 and most of it is original. I get inspiration from the 20 foot ceilings and hand made ornamentation everywhere you look. They simply don't build things like this any more. There are quiet reading rooms, large tables, plenty of light, and oh yeah, a Crazy Mocha coffee shop in the building. I use a cell phone dongle on my laptop and most people know that email is my preferred communication method.

If I need a break, I can look at priceless artifacts in the Carnegie museum through the windows in the open stacks. Or just get the world's most disgusting hot dog at the "O" a block away. If I need inspiration, that'll either make me or break me.

One of these days, I'd like to make the claim that some incredible technology of the 21st century was conceived in an edifice borne out of the some of the best technology of the 19th century.

"My aspirations take a higher flight. Mine be it to have contributed to the enlightenment and the joys of the mind, to the things of the spirit, to all that tends to bring into the lives of the toilers of Pittsburgh sweetness and light. I hold this the noblest possible use of wealth." - Andrew Carnegie at the Dedication of Carnegie Library of Pittsburgh, November 5, 1895.

Chapter 5

The Programmer's Lifestyle

87. What got you "hooked"?

Nothing interesting about my first story: I got a boring cubicle job in a large enterprise that needed 12 programmers to do anything, blah, blah, blah. But I did do some good work for a vice president who remembered me when he moved to another company, which leads to my second story, my real story:

He brought me in to his new company to do a consulting job to answer 2 questions, "What do we have to do to get Order Entry, Shop Floor Control, and Standard Costing written and running?" and "How many programmers do I need to hire?"

They had 400 employees, were missing all of these mission critical apps, and had only one programmer. But he was all they ever needed. I worked with him for 3 months and he wrote all of the software needed using tools and techniques none of us had ever seen before.

He did instant analysis and design, wrote disposable apps, did rapid prototyping, stepwise refinement, and extreme programming years before anyone ever heard of these things. He never wrote the same line of code twice, writing standard functions and reusable components. If he knew he needed something twice, he wrote a parameter-driven code generator on the fly and had me collect the parameters for him. He even coded in Boolean algebra and had an engine that converted it into production source code. He threw things into production long before anyone else would, figuring it was easier to just keep reworking them instead of waiting until it was perfect.

He was a one man shop in a $150 million company. He was smart, he worked hard, he loved what he did, but most of all, he didn't know that "it couldn't be done", so he just did it.

Three months watching Dick build stuff, and I was hooked for life. I had to do it, too. And I've been doing pretty much the same thing ever since, pausing only long enough to add a few new technologies to my tool box along the way.

Sometimes I wonder what horrible cubicle I'd be wasting away in if I hadn't met Dick and saw what was really possible.

88. What Matters Most to a Programmer

The longer I am a programmer, the more I realize that I love my job because of the actual work I do. Period.

I love the idea that people are trying to get things done, but need a little help from me to get them the tools they need. I love discovering with them what they need and how to get it to them. I love all that data sitting on disks somewhere begging to be used. I love all that data outside of any computer begging to be put on disk. I love the idea that I am master of a little universe that I can see in a 19" square right in front of me. I love manipulating important things, both complex and simple, with just little flicks of my fingers. And most of all, I love seeing something that came from nothing work for the first time. I did that! (Happy dance)

Oddly, not much else matters...

I have worked in the most deploreable conditions at the most difficult times and hardly noticed when the work was good.

OTOH, I have worked in Class A office space with the nicest people and best conditions and was ready to jump out of the window from boredom or frustration.

Yes, the more I think about it the more I realize that it's the "work" that matters. If it's important enough and I'm allowed to do it, I don't need much else. If it's not, then there is no perfume could that make that pig smell good.

89. Is programming hard work?

""He is dead, too much hard living!". Too much hard coding would be more like it."

Wow, those ones and zeros must have really been heavy!

Every time I see landscapers, construction workers, farmers, nurses aides, or anyone in one of my customers' factories or warehouses, I thank my lucky stars that I was born when I was, I had an aptitude and interest in programming, and I found the perfect career for me.

Sure I work hard, but my hard work is hardly the same as "their" hard work.

I've spent a career on my ass, building applications that hopefully make the lives of those who do physical work just a little easier.

My last career was a cook. After a 6 hour shift in a 110 degree kitchen serving 2,000 meals, a 12 hour stint in an air conditioned office in my Aeron chair seems like a vacation. Oh, and did I mention I earn more in a month than I did in a year as a cook?

I never expect my boss to thank me for anything. My boss (me) gives me the best bonus I could ever ask for. I get to do it all over again tomorrow.

90. Why I Do Not Feel Like a Fraud

There were many times when something I did seemed "too easy". So many times I'd listen to the user, understand their problem, and help them solve it with software. No big deal. Many of us have been doing that for years. Then the user would say, "Thank you, thank you, thank you!" "You're so much better than anyone else we've tried," or "This software is incredible! You should take it to market."

Or when I got compliments I didn't think I deserved. People called me "the smartest programmer I ever met," "brilliant," or "head and shoulders above the rest", and I knew it wasn't true, but accepted their praise anyway. And believe me, I'm not bragging, I'm just sharing experiences that many others probably had too. Others thought we were geniuses and we thought that we were just doing our jobs.

So why don't I feel like a fraud? Because I paid my dues. I may not the smartest or most gifted guy, and I'm certainly no genius, but I know I did the hard work.

While others were partying, I was debugging til 2 in the morning. While others were at lunch, I was eating a sandwich at my desk trying to find a better way to do something. While others promoted "good enough" software, I dug down 5 more levels to overcome the tradeoff. And while others accepted the status quo, I wrote my own framework to provide software that was in a different class from theirs.

I bet many of you share the same experience. Sure, we have been given gifts that many others never got, but those gifts only took us so far. We had to learn how to use those gifts. The real successes came from hard work. The kind of hard work that many people I know never do.

So if someone "overpraises" me, that's OK. I know I don't deserve it, but I accept it anyway. Kinda makes up for all those hours when I was slaving away and no one said anything at all.

91. How do I become addicted to programming?

I think that your first step in becoming "addicted to programming" is understanding what building stuff really is: intense sprints of orgasmic discovery separated by long periods of building the prerequisites. Once you understand which phase you're in, you can better understand why you feel the way you do.

A perfect example was my project last week. I had to make 14 changes (including 3 major structural reworks) to an existing application to produce one additional output. This output was key for a fundamental shift in thinking about the use of the app.

My week was actually fairly predictable:

Monday was fun for a while, thinking about how it would work, what needed to be different, how the data has to be restructured, and how cool the new results would be. But things became drudgery when I had to lay everything out build my work plan.

Tuesday sucked, tearing apart code while trying to maintain an audit trail and balancing the new functionality with keep everything old still working.

By Wednesday, I didn't even want to work on it and spent a lot on time on-line, playing games, and surfing the internet. I struggled getting the final pieces in place.

On Thursday, my interest piqued when regression testing went almost flawlessly. But no matter, even if it didn't, being in "getting it to work now" mode always captures my attention.

Friday was bliss. I was cranking code, unit testing, regression testing, reworking, getting the results I forsaw on Monday, and doing a happy dance every 20 minutes or so. Time flew as I worked into the night getting all this cool stuff working. I wish every day could be like Friday, but I needed to pay my dues earlier in the week just to get to that point.

I wish I had an easy answer for collapsing the long hours of building the prerequisites and expanding the time spent "in the zone". The only suggestion I can make is get to the point of having "something" for output as early as possible. That's when the fun usually begins. Until you reach that point, just understand you're in "courting" mode. Just keep on working and the digital orgasms will arrive. Once they do, it'll be worth it. Don't worry, you'll be "addicted".

92. Why do we lose our passion?

"So why is it that as we grow up we lose all the passion, the energy, the will and the strength to keep our dreams alive."

Because we lose so much energy in general.

Because we don't take care of ourselves.

Anyone who has ever been to any Tony Robbins event knows that he always starts with health (eating right, exercising, and mental attitude) before addressing anything else. Because he already knows what many don't: if you're not feeling right, nothing else matters.

Almost all of us have ample energy in our teens and twenties. But as we get older, we have to make a conscious effort to maintain energy and vitality. Most people I know in their 30s, 40s, and 50s don't do nearly enough. We slow down, put on weight, and lose energy. It happens so gradually that we barely notice. And we blame everything else: family, finances, lifestyle, etc. And we avoid acknowledging the elephant in the room: it's awfully hard to get passionate about anything when we only have enough energy to plop down in front of the TV with a bag of chips.

Most people I know with the energy to accomplish a lot, regardless of age, take care of themselves, especially those working 2 jobs or running a side business.

All of the other reasons (fear of failure, responsibilities, etc.) make sense, but I wonder how many people don't follow their dreams simply because they've run out of energy by not taking good enough care of themselves.

93. Why are languages so unimportant?

I am proficient in about a dozen languages, but only use 3. I go back to the SPS, PL1, Fortran, and COBOL days, and would rather chew razor blades than ever use them again. Today, I use javascript on the client and php on the server; I have yet to find a problem I couldn't solve with them.

In the years in between I discovered BASIC and have found it to be the Swiss army knife of apps. Not the old Dartmouth Basic, and certainly not anything Microsoft bastardized, but there are many other versions that seamlessly integrate with relational data bases that I think are a dream.

It's actually reached the point where I "think" in BASIC, design my app, and then sometimes write in in javascript or php.

I certainly wouldn't recommend my approach to someone else, but I think that's the whole point. Use the right tool for the job and get good at it. I see no need to learn new languages just for the purpose of learning something new. If I get "curious" or want to expand my mind, I can think of several hundred app problems waiting to be solved in any language, take your pick.

94. How does age affect programming?

Things I am worse at at 55 vs. 25: the 50 yard dash.

Things I am better at at 55 vs. 25: everything else.

Seriously.

Magic Johnson (great American basketball player) once described the difference of being a world class athlete at 30 vs. 20: you have to be a whole lot smarter about how you use your body and recover when competing with younger people.

We programmers on the other hand get the best of both worlds: the ability to work smarter against a physical and mental landscape that doesn't degrade nearly as fast as a world class athlete.

Frankly, when it comes to work, I can't think of a single thing I "used to do better", not creativity, not work habits, not personal habits, not physical or mental toughness.

I believe that programming, like running, is one of those things where the performance curve can remain flat until retirement, providing you take care of yourself. I'll let you know if it does.

95. How much does age affect ability?

I am 55 and absolutely do not care what anyone else thinks about it.

When I first started out I quickly rose through the enterprise ranks and was never taken seriously because of my age. As a wimpy looking nerd, I had always been underestimated by others and I found a way to use that to my advantage. When the time was right, I would just shoot them between the eyes with the right solution. My young age didn't matter.

Fast forward 30 years. I "never" notice age discrimination. It may be there, but I simply don't notice it. I think being in IT and in my 50's is a "tremendous advantage".

For every issue I have to address, I have that many more instances of experience dealing with something like that. Many more iterations of similar patterns to draw from.

IT is one field where you can actually get "better" with age. You don't have to run fast or carry heavy loads, but you do have to think nimbly and get things done, both of which get better with lots of practice.

IT is also one field where "what you get done" is more important than "who you are". This is always good news for us hackers and makers, regardless of age, sex, background, or anything else.

I am currently writing the best software of my life, by far. Not just "how" I'm writing it, but "what" I'm writing. I have seen so much that I have a natural instinct for what is needed, what works, and how to best go about it. New technologies keep me fresh and engaged. I feel perfectly at home here among younger programmers. I can't imagine a better place to be, with 21st century technology and 30 years experience!

To me it's odd that others in my age group don't feel the same way. Then again, maybe it's just state of mind.

My grandmother taught me one of the most important lessons of my life, "If you look hard enough for trouble, you'll

probably find it."

There is negativity everywhere, about ageism and a million other things. The secret is that it's only data to process as you choose. I have decided to ignore it and continue to do what I love and love what I do.

96. My Typical Day

I am running a marathon, not a sprint, so I frame my long working hours within a "healthy schedule". My typical day:

```
7:00 am - immediately start coding last night's plan, then email, headlines
8 to 9 - exercise
9 - breakfast & internet
9:30 to 12 - code
12 - lunch & internet
12:30 to 6 - code
6 - dinner with family (home or restaurant)
7 to 9 - code
9 to 11 - computer off, pencil/paper, analysis, design, detail plan for tomorrow
```

Exceptions: one to two days per week consulting (still keep my night schedule), one day per week with family, occasional sports on TV. I could keep this up forever.

97. Are you glad you became a programmer?

One of the best decisions I ever made.

Is is "what I thought it would be"? I don't know. Because I had no idea what to expect. (I did my first programming on the job; I started before there was much opportunity to do it on your own.)

I have done projects at over 80 companies. I have gotten involved in almost every aspect of the business. I have travelled all over the country, met many interesting people (and friends for life), and have constantly been learning and doing. Oh, and I have earned far more than most of the people I have ever worked with. It wasn't unusual for me to be earning more than my supervisor and much more than my users.

I have done lots of work on my own and have taken lots of time off between gigs.

Sure, there have been lots of negatives. I've even thought of leaving IT and doing something else. I know many who have. But then I think about it and realize that "this" is what I still really want to do.

There have been horrible working conditions, unreasonable people, terrible projects, long commutes, and worst of all, boredom and disapproval on someone else's project. But instead of whining, I always did something about it. I either fixed what was broken for me or moved on.

Because of modern technology and lifestyle, I am more excited about being a programmer than ever before. I don't want to sound like an old timer, but I clearly remember how hard it used to be to get good. I had to go to expensive seminars or to one of the half dozen good technical bookstores in the U.S. My first computer cost $6000 (double that today). Now with cheap hardware, google, downloadable environments, on-line forums, and Borders around the corner, everything is so easy! I just can't get enough.

For someone even mildly interested in programming, I would say, "Go for it!" Get a job and play around on your own. Learn as much as you can, technical and business, and if you don't like where it's heading, find a way to make it

work for you. Give it a chance. I'm sure glad that I did.

98. How much easier is it for an expert?

Layperson who's never seen it: "Impossible"

Practitioner who's becoming better: "Extremely difficult"

Expert: "We do this all the time. What's the big deal?"

I have reviewed or maintained the code of thousands of other programmers, and I've encountered maybe a couple dozen I'd actually hire and about 5 I'd consider as technical co-founders. What's the biggest difference? Until today, I wasn't sure how to verbalize, but now, I think a good description would be those who appear to take OP's advice and those who know better...

As far as I'm concerned, there's is a close correlation between good code and user experience. There's a close correlation between readable code and maintainable code. There's a close correlation between expertise and precision to detail "throughout". And perhaps most of all, there's a close correlation between something built properly to stand the test of time and "long term" user satisfaction.

If you want to learn a hobby, develop a passion, or really dig deep, by all means, just code it. Sometimes that's simply the best way to understand what goes on under the hood and learn what's possible once you learn the right way to build things. "Once you learn the right way to build things."

But please leave your experiments on your own hard disk where they belong. You may have thought that those bleeding pixels were cool, but your name will be cursed by the poor souls who forever have to maintain your mess.

99. The Introvert Factor

There's one huge factor at work, for many programmers. I'm tempted to call it the "wimp factor", but that's too negative, so I'll just call it the "introvert factor". I'm a perfect example...

I was always small for my age and looked nerdy with my glasses and attraction to books, etc. I was always picked last for sports teams, drew little attention from girls, and was usually the first one to be bullied. It even happened in my own family, subconsciously I hope. It was always easier to pick on the little guy to get what you want.

Fast forward to adulthood, and not much has changed, especially with bosses. It seems like my boss was always a sales/business guy, extroverted, and bigger than me. His/her natural reaction was to bully, probably because they knew they could get away with it. This was for almost everything: project management, discussions about work, and of course, money.

No more. I don't know exactly when it happened, but I decided not to take it any more. The more anyone picked on me, the harder I shot back, right between the eyes. Nothing pisses me off more than being bullied, especially about money.

100. What athlete are you most like?

Interesting question.

Reminds me of an argument between 2 great billiards players, Willie Mosconi and Minnesota Fats. Willie was the best player in the world at the time, winner of many championships. Fats was a hustler who never won anything except money. Each vowed to kick the other's butt in a match. Of course, Willie kicked Fats's butt, to which Fats responded, "So what. If there was money on the table I would have won."

This article made me realize I am more like Fats than Willie. I have interviewed and worked with many great programmers, but hardly consider myself in their class. They may know 27 ways to sort data, but being like Fats, I only need to know one to get the job done (and take the money).

If there was a competition, they would kick my butt. But if a client had to get something done, I could hold my own against any of them.

A typical scenario I have seen over and over: Competitor A is doing B, so we have to do C. This means that we'll have to modify our app to get data x from source y and present it to certain people so they have what we need to compete. We're losing $10,000 per day by not doing this, so we need it ASAP. My solution is rarely optimal, sometimes clever, sometimes not, and often a hack. But it gets done quickly, it works, and it "does the job".

That makes me more like Fats. But since I still aspire to be more like Willie, I'll start doing a few more things on these lists.

101. Do you need to "sprint" to get things done

Whenever I read a post about the sprint to launch, I think 3 things:

1. Great determination, great work ethic, great job.
2. It doesn't have to be this way.
3. It shouldn't be this way.

I feel fortunate that my DNA is blessed with some sort of internal "governor". I don't know where it came from, but I've always had it. Here's how it works: It stays out of the way when I am enthusiastic about something, allowing me work ridiculous hours and pursue almost anything that looks promising, whether it makes sense or not. But when I reach a certain point, it turns me off, completely. I don't seem to have conscious judgement of what that point is or when I reach it, but when it happens, I know.

A few examples:

- I have worked many times without sleep, preparing for a launch. Sometimes, I know my judgement is failing and continuing would cause more problems in the long run. So I stopped and apologize to everyone. I went to sleep and informed everyone that the project would resume at x. I'm not really sure exactly what happened, but I know I had little control over the governor.

- I had 2,500 invoices spread across the carpet, looking for a clue about a bug. After 8 hours, everything was fuzzy. So I just gathered up the invoices, filed them away, and went to sleep. Three days later a lightbulb went off, I spread out 100 of the invoices, and found the problem in 15 minutes. I know that if I had continued that night, I never would have found it.

- I worked 90 hours per week for 2 months for a big deployment. Without telling me, my co-founder spent all of our reserves travelling to a customer site to oversee the install. He emailed me every 20 minutes with a problem. Between being pissed off at him and exhausted from working on the wrong things, I realized the project was going

nowhere and would never succeed. So I just stopped working completely. I went to bed and didn't answer email for 4 days. I'm not proud of this, just one more story about my internal governor.

I'm a little frustrated that I don't have much control over my governor, but also a little relieved that it does it's thing. After all, I've never really been burnt out, and I'm still going strong. Thank you, governor.

102. Are young programmers better?

I've been programming commercially for 32 years and in all that time, I have found very little correlation between age and ability to deliver quality software.

I have worked with younger, inexperienced, and uneducated programmers who were willing to learn, with minds like sponges and who were a pleasure to work with. They often found or thought of things the rest of us overlooked.

I have worked with younger, inexperienced, and well educated programmers who thought they knew better and were obstacles to progress.

I have worked with older programmers with the same one year's experience 22 times. Oy.

I have worked with older programmers with excellent domain knowledge and limited technical range. Their personality and willingness to succeed were often the key to progress.

I have worked with older programmers with excellent technical range and limited domain knowledge. Sometimes it's hard to teach an old dog new tricks, but when you can, results can be golden.

I have worked with brilliant older programmers with extensive experience and open minds. The best of all worlds.

(By the way, I have also worked with programmers of many ethnicities, female, handicapped, gay, Republican, religious, even left-handed, and have found little or no correlation between their "description" and their "performance". One of the beauties of programming is that the easiest way to evaluate your performance is through your work itself and not much else.)

103. What's the advantage of working for someone else?

I have been in this situation many times before and always struggled with it. Until I figured something out...

"working in an internal IT department at a non-software company" can be a HUGE advantage. Why? Because your "customers" are right there.

Please do not underestimate this as part of your career planning. Sure, we all want to create cool technology, but As far as I'm concerned, the single biggest shortcoming for developers that I've ever seen is what I'll call "detachment from users".

You don't have that problem. You have end users right there at your fingertips. Take advantage of it! Learn from them. Practice your systems analysis skills. Find out how to make that missing link, the connection between technology and people, work properly.

I have written significant pieces of technology for software houses, but was actually "more" frustrated. Why? It took months, sometimes even years before anyone actually used it. And then, they were communicating with someone other than me. The technology was cool, but by not closing that loop with my customers, I always felt that my work (and learning) was incomplete.

By all means, continue your dreams and career plans. But don't waste your current situation by being depressed about it. I can't tell you all the times in my career when I stood head and shoulders above my peers because I had suffered in the trenches like you are now. Except you don't have to suffer. Because now you know to look for opportunities to get a lot better in other ways. They're all around you if you just look.

104. Why is it so hard to find programmers?

"Why is it so hard to find programmers? Are people afraid of joining a start-up?"

Let's not overlook the big differences between working at most start-ups and working at most companies:

1. Every programmer, no matter how good, is at least a little insecure. Every one of us doesn't know "something". Is the something you don't know going to make or break the next project? In a start-up, there's rarely a safety net to catch you, but in a larger company, there's probably a better chance that someone else can help you along.

2. It takes a special mentality to work in a less certain environment. This is more a matter of personality than skill. My mentor was fearless. He used to say, "I didn't know I couldn't do it, so I did it." This attitude, as much as skill, determines how well one would thrive in a start-up.

3. What happens when things go wrong? (And they will go wrong.) The ability to recover from problems in a larger company is a great asset. In a startup, it's a necessity. I've met many enterprise programmers who could crank out great code between 8 and 5, but melted under the pressure of an all night emergency. They would never survive in a startup.

4. What happens if you don't feel well or if your mind is "someplace else"? In a larger company, you could coast for a day or two (maybe more). That's rarely an option in a start-up; time lost is time lost forever.

5. In a larger company, you can do quite well whether you have deep domain knowledge or you're a jack of all trades. In an early start-up, you better be both.

6. Ever wonder why waterfall development refuses to die, even though it's not as effective? Because so many of us have to have a road map in order to function. "Road map" personalities don't fare nearly as well in roadmapless environments (many start-ups).

7. A start-up programmer must have at least a little maverick blood. If you believe everything you hear and do everything everyone else is doing, how can you differentiate yourself? In a larger company, you may not have to. In a start-up, you probably do.

8. Is there something you simply have to do? Then you probably belong in a start-up environment. It's tough (although not impossible) to get the same opportunity in a large company.

9. Do you think the work is really cool? I know lots of good enterprise programmers, but have trouble thinking of very many who think their work is cool. They like their jobs, but work is "just a paycheck". Not the type of people who would thrive in a start-up.

10. Do you do a happy dance whenever something works for the first time? Then you may be more comfortable in a start-up than in a big company.

105. Is there anything good about my job?

What's good about it?

What's good about it?

What's good about it?

(I had to ask multiple times because we all know that the first couple of answers would be, "Nothing".)

Every job, no matter how boring, is loaded with "stuff" that you "can" use to contribute to your long term progress.

It may be access to a user who's an expert in their field and would love to share their expertise.

It may be a project that needs to be done, but no one else has time. And you can learn a lot of unexpected stuff from doing it.

It may be lots of interesting data on their hard drive that you can learn a lot from just by transversing and/or organizing.

It might even be proximity to a quiet coffee shop where no one would miss you.

It could be anything.

So if you're stuck, then it's your job to turn lemons into lemonade. (Practice turning lemons into lemonade is an invaluable skill on it's own; just ask any entrepreneur.)

Now close your browser and make a list of 10 things you can try to get "some" value out of this job while you're there. Then open your browser back up and let us know what they are.

106. The Performance Economy

As a young geek, I was often passed over because I didn't "look like" I could do the job. It started with little league and continued on through high school and into the work place. I even quit one of my better jobs because a slicker looking but far less talented peer was promoted into what "should have been" my job.

No more. As a mild mannered adult geek, I still get the same pseudo-negative first impression reaction until I open my mouth and show them how much they'd benefit from me and technology.

The Performance Economy was a long time coming (and probably taken for granted by anyone under 25), but now that we're there, there's no going back.

107. Becoming Senior

There's a chasm you have to cross to become "senior", whatever that means. One of the biggest skills you need to learn is not technical. That skill is understanding exactly how to apply our limited resources.

Should we worry about how well a certain module was written? Maybe, maybe not. Sometimes the answer really is, "It runs well and isn't hurting anything, so let's leave it alone. Opening up that can of worms dumps 47 new issues into our queue and now is not a good time to do that." Or the answer may be, "This has to be made right before we can add these 27 other things to the system or we'll all be in deep sh*t." How do you know which? Experience. Understanding the big picture. Understanding your customers and users. Understanding your dev team. You get the idea.

Taking on a lead role is not selling out if it's used to expand your horizons and make you an all-around better dev person. When you return to full time development, you'll be light years ahead of those who never had to manage the whole project. A little perspective goes a long way.

108. What should an older entrepreneur do?

Pair their work ethic, real world experience, and life lessons with the passion and technical skills of a 20 something hacker.

I started my first business when I was 27. My partner was 41 and had done things I hadn't even imagined. He was so smart, so seasoned, and knew the ropes about so many things that he saved us both countless hours and dead ends. And I was able to do things he never had a chance to learn. We made a great team.

Now I'm on the other side of that relationship. And would love to do it again with someone in their 20's. I have a million ideas that come from years of real world experience and not enough time to act upon them.

109. What's hardest about programming?

What a solitary task programming is.

This is the hardest thing for me to explain to others. And still one of the hardest for me to get used to myself. It takes a lot of time working alone to get anything done.

It may also be one of the many reasons Hacker News is so popular. I don't know about you guys, but if I didn't have this place to break up the loneliness, I'd probably go nuts.

110. Where did you learn what you need to know?

Where did the things I need to know to do my job come from?

Mom, Dad, & family	10%
kindergarden	5%
elementary school	1%
middle school	1%
high school	1%
college (B.S. math)	1%
college fraternity	5%
business school (MBA)	1%
my first mentor	5%
my second mentor	10%
my users over the years	10%
my employers & customers	10%
reading	5%
other programmers	5%
doing on my own	30%
	100%

111. Why didn't you pursue mathematics?

I left math for a totally different reason and I'm almost embarrassed to talk about it. A little background...

I worked full time through college and graduated with less than $100 in the bank. I had opportunities to go on to graduate school for either math or business.

Every professor in our math department drove an older subcompact except the department head who drove a Chevy Impala. Imagine, work your whole life, get to the top of your field, and drive an Impala!

I had struggled too long to set myself up for more struggle. So I went on to get my MBA, learned how to program, and have never had a lack of good quality, high paying work. I'm a little embarrassed that I was so shallow back then, but maybe my subconscious was trying to tell me something. I love math, but I'm sure glad I made the choice I did.

At a recent math reunion, I felt right at home once again. I met a buddy who graduated with me and continued on to become a tenured math professor at a major university. I asked him how he felt about his choice. He told me, "I'll never be rich, but I teach calculus for 8 hours per week 9 months per year, I don't have to publish, and my wife and I have visited over 100 countries. Not a bad life at all."

112. Should I keep my day job?

Yes. Here's why...

You get requirements from your daytime job.

This is so important that when I stopped moonlighting and went full time on my start-up, I kept an outside client 2 days per week. Let me explain...

One of my biggest problems has always been WHAT to build, not HOW to build it. Many successful start-ups have been the result of building what one needed oneself. I just expand "oneself" to include my parttime clients. Think you have needs? Wait til you see the long lists of needs in almost any small business. Sure, I lose a little time in the development cycle, but I more than make up for it at user acceptance time. No cycles there - I already know what they want.

113. How do you balance work with your SO?

If you "really love her", then you already oughta know what's most important. (Most important, not the only thing.)

Her "expectations" are not really expectations. They are cries for help. Listen.

Some of the things we have done:

- We take French lessons "together" and speak French around the house all the time.

- We always eat dinner together.

- We go out somewhere every weekend (usually not my pick).

- We catch up by phone several times a day.

It's hard to explain, but I have a "more balanced lifestyle" "without" "lower expectations". You "can" have both. It's up to you to find a way.

[EDIT: Forgot one of the most important things: Sunday brunch is a big deal at our house. We shop together (the night before), cook together, and sit together for 2 to 3 hours at a table full of great food and 3 newspapers. At first, I thought it was a waste of time, but now I really look forward to it.]

114. Have you ever been burnt out?

I have been programming continuously for 30 years and I've "never" been burnt out. In fact, I'm having more fun than ever. I can't imagine doing anything else.

I have worked in 88 companies, either as a contractor or an employee. I have seen other scenarios play out many times. I have worked on the worst garbage code and dealt with incredibly nasty and incompetent people almost everywhere I've been. But I never let them beat me.

Only I get to decide how I feel about anything, especially work. When things have gone sour, which they almost always do, I have done everything I could to fix them, and when that wasn't enough, I have moved on. I've always felt that one of my biggest strengths was the breadth of my experience. I use almost all of it every time I do something.

There's now more hope and opportunity for true hackers than ever. It's just a matter of continually finding your best place. Maybe a small service business. Maybe that perfect job. Maybe a startup. If all else fails, keep the lousy day job and hack away at something cool at night.

If you're really burnt out, take a break. But make it your decision - don't let the bastards take your inner hacker away. Ever.

115. Why were you such a late bloomer?

"1955 was the best year for geek births"

For outliers, maybe, but for the rest of us, NO!

I was born in 1955, on the same day as James Gosling, inventor of Java.

But I was not an outlier. I was pretty much a regular person from a middle class family who went to public school and aspired to be the first in my family to graduate college. Here's the problem with being born "too soon":

I graduated high school, college, and graduate school "without ever having touched a computer". Think about that. Neither of my colleges even had a Comp Sci department. I grew up in Western Pennsylvania, far from the leading edges of Boston and California. Nobodyknew anything about computers. They were giant machines that sent your electric bill. Period. Unless you were fortunate to be close to the geek counter culture of Northern California or had parents with millions of dollars, forget it.

I graduated with an MBA and got a job as a restaurant manager (1978 was a lot like 2009). Then I picked up a COBOL book and practically memorized it for a difficult to get programming interview. I got the job and the rest is history.

I often wonder what my life would have been like if I had been born 10 or 20 years later. But it's a wasted thought. What if I had been born 10 or 20 years earlier. I'd probably be a retired car dealer now.

The great thing is that now it just doesn't matter. I took a while to get here, but I can't imagine doing anything else. Maybe that's why I'm here so often: making up for lost time.

116. What do your parents think you do?

Dear Dad,

My customers are small business people (retailers, wholesales, doctors, lawyers, etc.) who own computers that have replaced their file cabinets and some of their clerical employees. Those computers came with lots of stuff in them but need more as their business changes or they discover stuff they forgot. I upgrade their computers with the stuff they need. We call that stuff "software". They pay me. Well enough for me to buy you dinner Sunday night and take you to the Steeler's game. What do you say?

Love, Eddie

Dear Mom,

I sit in an office writing all day long. I have a fridge and a microwave and occasionally go out to lunch with people down the hall. When it gets cold I wear the sweater you bought me last month. I love what I do. I write stuff, kinda like Stephen King or Danielle Steele, but business stuff, not fiction. My customers love what I write for them and they pay me well, so you never have to worry about me again. I showed Uncle Lenny what I was working on and he thought it was great. I'll pick you up for lunch and a trip to the mall at noon on Saturday. See you then.

Love, Eddie

117. Why are some programmers so condescending?

Condescending feedback says more about the speaker than the listener. It is almost invariably about their own insecurity. This is true is almost all fields of endeavor, not just programming.

Just a few examples of my own:

Insecure bridge player: The queen of spades was a stupid play. What's wrong with you?

Excellent bridge player: The queen of spades would have been a great play against a 4/2 split. But since you had a 3/3 split, what do you think would have happened if you had played the ace instead?

Insecure public speaker: You look like an idiot playing with your hands like that.

Excellent public speaker: You talked about a lot of cool things. I bet I would have been even more interested if I wasn't distracted so much by your hand gestures.

Insecure parent: If you can't keep that baby quiet, you should just stay at home!

Excellent parent: Here's something that has really worked well for me when my kids cried in public...

Insecure programmer: How lame. I can't believe you .

Excellent programmer: I see that works. I have found a few ways to make it work even better. Let me know what you think.

118. What are the biggest geek myths?

"1. Recognize that people will know you are a geek from the moment they meet you"

Assume nothing. If you're not sure about something, ask. Give the other person the benefit of the doubt at least once.

"2. Don't try to change people's preconceived notions of geeks"

Don't try to change anything about anyone else. Just be yourself and engage them.

"3. Don't get too comfortable and start being yourself"

Always be yourself. Who else are you going to be? And who is going to be you?

"4. Try to talk as little as possible, and when you do speak, only ask superficial questions"

Take advantage of this excellent opportunity to engage with other people. Learning is maximized for everyone when all talk and listen.

"5. But don't ask questions about things that normal people should know"

How else would you know what's "normal" unless you ask?

"6. Temporarily let go of the urge to achieve absolute precision in speaking"

Sometimes absolute precision is exactly what's needed to improve communication. The trick is to know when. Learning when comes from practice.

"7. Don't correct anyone even when they're incorrect or imprecise"

Again, the trick is in judging context, which comes from practice. If they said they did something a million times, obviously no correction is needed. If they're giving instructions on defusing a live bomb, then you better correct them.

"8. Don't use words that an 8th grader doesn't understand"

Again, how would you know? Be yourself, say what you mean, and learn from the feedback.

"9. If somebody asks you about your job or hobbies, answer in one sentence"

Answer in as many sentences as you deem appropriate. You're probably a pretty smart person. Exercise you judgement, which will become stronger just as if you exercised your biceps.

"10. If everyone around is enjoying the ambient music, background live performance, etc., don't jump in with any analysis"

Why not? Sometimes the most interesting conversations get started this way. Again, your judgement is way more important than OP's rules.

"11. Never start a sentence with "Did you know that ...""

Sames as #10.

"12. Never start a sentence with "You should really ...""

Probably better stated as, "Only give advice when it's asked for."

I'd prefer this simple list of social tips:

1. Be yourself. Being rejected by someone else for being yourself is a self-correcting problem. They just saved both

of you lots of time and energy.

2. Treat others how you'd like to be treated.

3. If you spend lots of time alone, take advantage of an opportunity to be with others by engaging and learning.

4. Use your best judgement (That's what it's there for.)

5. Have fun.

6. Take any list of rules with the word "actionable" with a grain of salt.

Chapter 6

Philosophy

119. Who is a superstar developer?

A smart accountant once told me that the answer to "How much money did you make?" is always, "Who wants to know?" If it's an investor, the answer is "A lot." If it's a customer, the answer is "A little." If it's the IRS, the answer is "None."

Same thing here. The answer to "Who is a superstar developer?" is always, "Who wants to know?"

To a project manager, the programmer who hits every deadline (regardless of quality) is a superstar.

To a customer, the programmer who solves their problem quickest is a superstar.

To a business owner, the programmer who makes them the most money is a superstar.

To a PHB, the programmer who makes them look the best is the superstar.

To a journalist, the programmer who tells the best stories is the superstar.

To a junior programmer, the best mentor is the superstar.

To another programmer, the programmer they are most likely to want to go into battle with is the superstar.

120. Do you think you have peaked?

Not even close.

30 years of programming and my best code has always been my most recent. I just keep getting better and better with no end in sight. And I love it.

As far as I'm concerned, there's absolutely no reason to believe the hypothesis of peaking and declining. Not me, that's for sure, and not most of the people I know in their 40's and 50's.

In this way, we are "not" like athletes at all. I remember watching Michael Jordan when he played for the Wizards, riding the staionary bike to keep loose during timeouts. What a pity, I thought, to be at the top of your game, then a shadow of yourself just 5 years later.

We programmers are not like that. Just take care of yourself and you don't have to level off until you would have anyway. All the new technology and opportunities keep the field fresh. And the people, too.

121. Why do you program?

I'm not in it for the money. The money is a barometer of something else. If someone can make money in this business (not that hard to do), then something they wrote is successful at some level.

I can confidently say that 90% of the code I have ever had to maintain is total garbage. I am often stunned that it even runs and I used to wonder how it ever made it into production. (I don't wonder any more, now I know, hardly anyone QA's source code anymore.)

(Another example: Today I refactored 1200 lines of code that edits credit card numbers down to 46 lines, removing 4 bugs in the process. This has been running in production for 18 years. This is not a joke.)

I realize that what I see is not a random sample of all code. I never get to work on the good stuff; no one does - that's the whole point.

My work, on the other hand, has been wildly received by my customers and users for many years. I love what I do and love providing value to others. I can't imagine doing anything else.

For some reason (my internal metaprogams or something), I am much more highly motivated by seeing someone else's garbage doing well and thinking, "I can do WAY better than that," instead of people saying, "That was good." I don't know why.

Also, it's my experience that most people do not see the possibilities for truly excellent software (present readers excluded, of course). They believe they have to put up with my previous example. I can't wait to show them otherwise.

122. What are your hardest learned lessons?

A few of my hard earned lessons:

Time spent working does NOT necessarily = amount of work completed. It's easy to get discouraged when you "sense" you're not spending enough time on something. Two days ago, for example, I didn't work much. But it was a very productive day! By the end of the day, I figured something out and the flood gates opened. That's just how it works for hackers sometimes. If you don't care anymore, that's one thing. But being frustrated about time spent is a signal something else is wrong, not your commitment.

Some experts think we are either preprogrammed with "moving toward" or "moving away" internal metaprograms. OK, whatever. I'm a very positive optimistic person, so I just assumed I was a "moving toward" person. Wrong! I am HIGHLY motivated by that which I do not like, but I never realized it. If I see something I like, I think, "that's cool". But if I see something I don't like, I think, "That sucks. I can do way better than that." So I do. It may be one of the 7 deadly sins, but jealousy is a great motivator for "moving away" people. I'm older than every speaker at Startup Weekend, the Chief Justice of the Supreme Court, and our president. But I'm just as smart as any of them, and I haven't got mine yet. That just pisses me off! (See, it works, I'm ready to hack right now.) What about you? How are you programmed? What can you do to stoke yourself (or piss yourself off)?

"When do you know its time to call it quits?" Never. I realize most people will disagree with me. Walt Disney and Colonel Saunders went to a thousand banks before getting a loan. How soon would you have given up? When do you quit trying to teach you child how to walk because it's not going as well as you'd like? Jerry Seinfield jokes that it took him 20 years to become an overnight success. I've seen the same thing with many hackers I know. If you know deep down inside that your project is a winner, you must do whatever it takes to get it to "walk". If you're not sure, then maybe you shouldn't have started it in the first place. Only you can answer that one.

123. What have you learned from mentors?

The most important lesson my mentor ever taught me:

We went to a client to work on Problem X. He quickly determined that solving Problem X would achieve nothing. Problem Y was the real problem, but was way outside our areas of expertise.

So what did he do? He slept 4 hours a night for the next 2 weeks studying everything he could find about Problem Y. He reviewed reports, industry literature, called experts, and talked to as many people in the company who knew anything about that subject area. Within 2 weeks, he presented a brilliant solution that no one had ever considered but was instantly understandable by their experts. (That solution included work done by us and we had a great client relationship for years.)

Later, I asked him why he tried to accomplish something so difficult with such a seemingly tiny possibility of success. I'll never forget his reply...

"I didn't know that I couldn't do it, so I did it."

124. Who are the real heros of programming?

My customers.

My customers do great things. They often need my software, built and functioning properly for years to do these things. I love building stuff, but they are the real heroes. Just some of the things that they do:

```
- get the right drugs get to the right people
- get the ambulance to the right address
- get the right materials purchased and delivered
- get the right product built, on time and budget
- get the right product shipped accurately and on time
- make sure the parts going into that airplane are certified
- make sure your insurance claim gets processed properly
- make sure they make enough $, so they can keep doing it
```

I can go on and on, but you kinda get the idea. I love to learn, to optimize and refactor, and to build beautiful things. But what I do pales in comparison to what they need to do. I never forget that.

125. Relentlessness

I try to approach not to change the world, not to build cool stuff (well maybe just a little), but to genuinely help people. For a business person, this thinking is difficult and counter-intuitive.

Why do I do this? Because of my first mentor (and co-founder).

He was relentless in everything he did. I learned to stay up all night, keep calling on customers, and stay with tasks until we got somewhere with them. I remember many nights with thousands of invoices spread across the carpet, watching the graveyard shift run their machines, or scanning reports on-line, looking for clues. He wouldn't quit and the reason was always the same, "These people need help and we can help them. So we do. Don't worry about how hard it is or how much time we spend, it'll all work out in the end."

Sometimes I think that this is the attitude very successful people must have. It's too easy to give up when it's for ourselves, but much harder when we know that someone else needs us to get the thing done.

126. What if I'm not as good as someone else?

"If you're capable of writing the best web framework in the world in your spare time, chances are you can also create a business at the same time."

Don't make the mistake of underestimating yourself.

I'm not suggesting that you'll go out and write Rails in 3 weekends. What I am suggesting is that the more I meet "famous" hackers and the more I meet people from this community (online and offline), the more I realize that there's not really all that much that separates us.

Lot's of people are obviously brilliant. And even for those who are a little less brilliant, brilliance is only one part of the equation. Work habits, determination, perseverence, passion, and maybe most of all, belief, are just as important. Don't sell yourself short.

I have no idea if I am as brilliant as others. Odds are against me. But he inspires me to achieve things just as cool as theirs. And I just know that I can. I suspect most people can too.

127. It's Never Too Late

```
Teen years - flipped burgers & partied
Age 21 - graduated college, flipped burgers, & partied
Age 24 - touched my first computer
Age 25 - wrote my first program
Age 27 - touched my first PC
Age 31 - wrote my first low level code
Age 32 - started my first business
Age 39 - started my second business
Age 41 - accessed the internet for the first time
Age 44 - wrote my first browser-based app
Age 51 - found Hacker News
Now    - having more fun than ever
```

It's never too late, you're never too old, and it's not whether the glass is half full or half empty.

It's about getting up off your butt and filling the glass the rest of the way.

128. It Can't Be Done

"And in my experience when enough people are saying that 'you can't do that' there is an opportunity waiting for you that is proportional in pay-off to the number of people asserting that it can't be done."

Great thought.

Most of my most memorable successes were when others said that something couldn't be done. First you think, "Why not?" Then you think, "What would it take?" Then you figure that you'll never find out for sure unless you try. The reward is compounded by the initial skeptism.

Just a few silly examples (any of these sound familiar?):

```
Manager: Shop Floor Control is impossible.
Me:      Why?
Manager: Because the base data is so inaccurate.
Me:      So?
Manager: It would take years to fix all the data.
Me:      What if we turned in on anyway?
Manager: The output would be worthless.
Me:      Wouldn't it show where the base data was inaccurate?
Manager: Yes.
Me:      Then you could fix the biggest culprits?
Manager: I suppose.
Me:      So turning it on would expedite data fixing?
Manager: Yes.
Me:      So it's not really impossible?
Manager: Well...

Manager: Bug free software is impossible.
Me:      What would it take to make is possible?
Manager: Nothing.  Can't be done.
Me:      What if we added systems testing to unit testing?
```

Me: And then built rigorous test plans covering almost everything?
Me: And then enforced User Acceptance Testing?
Me: And allowed nothing into production without passing?
Me: Would it be better?
Manager: Yes, but we can't afford to do all of that.
Me: So, bug-free software isn't impossible, just expensive?
Manager: No, it's impossible. Get back to work.
Me: Sigh.

Manager: A web app is impossible.
Me: Why?
Manager: Because it depends upon data entered by regular people.
Me: So?
Manager: People are idiots. They enter wrong data all the time.
Me: What if we trained them?
Manager: Impossible. They don't work for us.
Me: What if we made the software smarter?
Manager: What do you mean?
Me: Data validation.
Me: Data reasonableness based upon rules or history.
Me: Crowdsourcing data validation.
Manager: The data would still be bad.
Me: What would it take to make the data good?
Manager: Nothing. Impossible.
Me: Sigh.

129. How perfect do you have to be?

I can attest that trying to hit a homerun and hitting only a single or double is still great.

I tried to prove Ferman's last theorem (years ago) for my senior project. I didn't:-) But I turned in what I had and got an A+ and 4 invitations to grad school based on that paper alone.

I entered my fraternity into a contest for Chapter of the Year. I didn't care about the trophy; I only cared about what would happen to us by doing all the things needed to try for it. It worked.

I have often tried to solve the biggest problem at some of my customers over the years, but didn't. But all the little things I had to do to make the attempt turned into pretty good products and services anyway.

Nothing wrong with trying to hit a homerun as long as you keep your wits about you and don't let it become and win or go home proposition.

130. Are good programmers born or made?

Made.

Smart. Motivated. Works well with others. Has passion. Good problem solver. Detail oriented. Able to focus.

In order to be a great programmer, how many of these are important? All of them.

How many are necessary? None of them.

I once had a calculus professor who said, "Many students are simply unsuited for the sciences."

I disagreed with him then, and after many years of work experience, I disagree with him more than ever. I firmly believe (with some exceptions) that almost anyone can become great (or at least very good) at almost anything. I've seen it over and over again.

I have worked with hundreds of programmers over the years and screened thousands of others.

Almost all of them consistently delivered substandard work. Not because they weren't smart or motivated or capable. More likely because they weren't taught properly and were in terrible environments.

Teach someone how to do the job properly, give them an environment in which they can thrive, give them a chance to do quality work, and treat them like human beings. Then watch what happens.

But companies are too stupid or lazy to do this, so they think they'll just hire talent and dump them into their already sour environment. Fix the environment and let regular people become great.

Made.

131. What's your greatest life lesson?

In the past year or two, I have learned my greatest life lesson. As a lifelong high achiever, it was extremely counter-intuitive yet it was right in front of me all along. First, a little background...

In the past couple of years:

- My father died.
- My aunt (and best friend) died.
- My cousin (who was really like my brother) died.
- My 19 year old cat died.
- We had our first ever family reunion.
- My mother's dimentia has turned her back into a child.

Sure we all have great memories and are busy working at building even better futures, but ultimately it all boils down to:

All we have is now.

My pets have been trying to teach me this for years, if only I had listened. And now my mother is teaching me. They don't really remember yesterday. They don't care about tomorrow. But they really care about the moment. Intensely.

I have had to really slow down and let this sink in. When I visit my mother in her nursing home, we have a great time laughing, talking, visiting others, and of course, playing Jeopardy. We can't have the conversations we used to, so we just have new experiences, one time only, in the moment, and only for those who are there. We never talk about the past and she simply doesn't understand, "I'll see you tomorrow."

I haven't stopped building my future, but I no longer sacrifice the present in order to get there. I have learned that the process must be as enjoyable as the outcome. After all, the process is "now" and the outcome is just an instant in

time.

It may sound cliche, but everyone should take inventory of all the good stuff in their lives (especially other people) and make the most of it "now". You'll be surprised how quickly it'll be gone. Don't wait half your life to learn my most valuable counter-intuitive lesson.

132. Are programmers expensive?

"Hardware is cheap, programmers are expensive."

"Mediocre" programmers are expensive.

Good programmers are the bargain of the century.

If companies would just wise up enough to pay a good programmer 3 times as much as a mediocre programmer to do 10 times the work, do it right, and do it so that it can be maintained reasonably, any tradeoff would become moot. But companies generally don't do this, which is probably one of the main reasons the best programmers go off and do their own.

133. How far from shore are you?

I have 2 signs above my desk.

One says, "It doesn't matter". This is for when I get so stressed out, I have trouble doing anything. It helps keep things in perspective.

The other says, "Jabez Wolffe". His guide boat forced him to abandon his swim across the English Channel because they couldn't see through the darkness and fog, and it was too dangerous to continue. What they didn't realize was that they were only 100 yards from shore, but they had no way of knowing.

How far from shore are you?

I would suggest doing whatever you can to find out before making any decision.

134. What is "fear of failure"?

I had a friend in college (I'll call John) who would shoot hoops, play golf, or play table tennis with anyone at any time. But he would never play anything else. He wouldn't play touch football, softball, bridge, or even shoot a game of pool. I could never understand it until I finally figured it out: he wouldn't play anything unless he knew that he would win. How sad, I thought.

I just realized (to my horror) that years later, I am just like him. I don't push boundaries like I used to. I don't call on that extra customer, volunteer for that project, or apply to programs like yc if I think there is any chance I won't win. There's always a reason: the software is missing too much, the demo sucks, there are 14 other things that have to be done first,... You get the picture.

I never thought of this as "fear of failure". I just got so used to succeeding in everything I did that I didn't want to do anything else where I didn't succeed. I became John without even realizing it.

I've got to change this stinkin' thinkin'. A good failure would probably do me good. Or maybe I should just try something I would have never imagined a month ago.

135. The Code is the Star

If you want to be famous, go be an entertainer, athlete, or politician.

If you want to be a programmer, check your ego at the door. The two biggest roadblocks to success in programming are incompetence and attitude. BigEgo = BadAttitude.

I measure my success not in fame, but in the value gained by those who use my software, and the value gained by those they serve, and so on, and so on. I don't know them and they don't know me, but I'd like to think the world's a better place because of all the ones and zeroes I've arranged. They are the stars and that's good enough for me.

136. How can I be excellent with a day job?

"...how do I get my mojo back and get that level of technical excellence back?"

You decouple your day job from your need for technical excellence.

You do something on the side. Maybe a pet project. Perhaps a little service work for customers you find. Contribute to an open source project. Or best of all, start your own business.

This is what I did and it changed everything. I have never complained about the lack of stimulation of any day job I have had (well maybe just a little). Better yet, I have used to crappiness I encountered during the day to push myself to "never do that" at night.

The day job is comprised of quality right in the middle of the bell curve and it's good enough to pay the bills. The side work gives me a chance to push all the way out to the right hand side of the bell curve with cool stuff.

The ultimate plan is for the side work to take over and make working on someone else's crap during the day unnecessary. Give it a shot.

137. How do you put your skills to good?

I've always thought that the best way to put your technical skills to the greater good is through your day job, not instead of it.

Some of my days jobs have been to write software to ensure that:

```
- people get the right prescription medication on time
- firetrucks and ambulances get to where they're supposed to be
- parts that go into cars and planes are properly certified
- prisoners are kept in jail
- those same prisoners get proper medical care
- electronic equipment gets assembled properly and on time
- medical supplies get dispatched to where they're supposed to
- insurance claims are processed properly
- quality data is properly maintained for food items
```

You don't need to do charity work on the side in order to contribute to the greater good.

On the other hand, if you don't think that the work you do during the day contributes to the greater good, then maybe you should consider doing something else with your valuable time.

Do good and get paid. You can do both at the same time.

138. Issues vs. Details

I have a simple guideline for real life interactions with others that carries over quite well on-line, "Deal with issues; ignore details."

It's amazing how well this works in person, especially when trying to get something done. My number one question to another is probably, "Is that an issue or a detail?" We can almost always decide together which it is. Then, if it's an issue, we deal with it, and if it's a detail, we move on to the next issue.

This has also saved me countless hours and aggravation on-line. If I post something and someone disagrees, I quickly decide whether or not it's really an issue and only engage the other if it is. I realize that this is just a judgment call, but I'd estimate about 90% of on-line disagreements are just details. In these cases, I think it's best to simply move on.

139. How is an issue different from a detail?

Examples are everywhere. In fact, almost every human interaction is an example. Here are a few off the top of my head:

Quality control rejected one program because it was indented 4 spaces instead of the standard 5, but accepted another, even though it had enough memory leaks to crash the server under certain conditions. The first was a detail; the second was an issue. It took me 2 days to get Q.C. to understand the difference.

A friend recently arrived for a dinner party an hour late and then complained to me that another spoke with her mouth full. As far as I was concerned, the first was an issue and the second was a detail. My friend thought otherwise about both.

Accounting recently spent 3 days implementing a new key policy for the private rest rooms (presumably to prevent theft) and then wrote off $50,000 of inventory because no one could find the proper paperwork. IMO, the former was a detail upon which much time was wasted and the latter was an issue that never actually got dealt with.

We spent the first hour of a recent meeting trying to determine naming conventions, but ran out of time before we decided if the customer's credit limit should be split between 2 divisions. Again, wasting time on details and not dealing with real issues. (This is a great example. One of the best ways to lose your shirt is to not deal with credit/collection/accounts receivable issues.)

140. Living in Two Worlds

Sometimes I think I'm living in two worlds, the customer world, where everyone is scrambling to get stuff done, and the startup world, where everyone is talking about what the customer world should be like.

Don't misunderstand me, though. I love the startup world. There is an underlying current of optimism that I rarely see in the customer world, where people are just too busy to see the possibilities if they hit them in the nose. Sometimes I have to grab my customers and yell, "Let's slow down for 5 minutes and think about a better way to do this!"

In the startup world, it's often too easy to lose sight of the definition of success. Success is not starting a business, getting into an incubator, or securing funding. Success is satisfying paying customers over and over again.

In the past 6 months, I've had 5 different customers ask me for the same thing. I described my approach to satisfying them to a startup investor acquaintance. He told me that no one would ever pay for it. Now I know I'm on to something.

I like living in 2 worlds. It helps me maintain perspective. Sounds like it works that way for you too.

141. What are the biggest programming myths?

"1.- We're always wrong."

SometimesWereWrong + SometimesWereRight != "We're always wrong"

"2.- If something can break, it will break."

Good thing this isn't true. I'm am continually amazed that some of the crap I have to maintain ever ran in the first place, much less that it continues to run. But it does. Sometimes for years without ever breaking. I call it "dodging the raindrops".

"3.- All code is crap."

Whoever said this must have never seen really magnificent code for him to say this. What a shame. It's out there.

"4.- There is always a bug."

False. The first step in writing bug-free code is believing that it's possible.

"5.- The most important thing is the client."

Lots of things are important, but it's hard to argue with this one.

"6.- Design on paper doesn't work."

Sure it can. Just because it usually doesn't work doesn't mean that it can't. I prefer prototyping, but blueprinting can be effective too.

"7.- Less is more."

Generally, yes. That's what great design and refactoring are for. But there are counterexamples to this.

"8.- Coding is only 20% of what we do."

If you actually believe this, then you're not spending enough time coding. I'd say more like 50 to 80%.

"9.- The customer doesn't know what he/she wants NEVER!."

False. The customer often knows "exactly" what he/she wants, but may have trouble communicating. That's when you expert analysis and prototyping skills come in handy.

"10.- Someone has done it before."

Similar to #4, the first step in inventing something new is believing it's possible. Lots of stuff that needs to be done hasn't been done for all kinds of reasons. Maybe no one thought it was possible or no one has understood its potential. But we know better.

"Bonus: Hey! Our job is cool!"

Yes! I agree!

142. Don't Pull a Teddy Roosevelt

Five minutes after winning the presidential election of 1904, Teddy Roosevelt vowed not to seek re-election in 1908. Six minutes after, he regretted what he had just said. He tried to return in 1912, but failed and regretted his hasty decision the rest of his life.

Why do I mention this? Because whenever I feel like I'm in a difficult situation (not all that much different from yours), I promise myself not to pull a Teddy Roosevelt and do something hasty that I'll regret forever. Neither should you.

For what it's worth, time is not slipping away. In spite of what you may think, 30 is not old.

My suggestion: keep your job and stay on your path, but find a way to do it and your startup at the same time. You have to get creative. Put in a few hours on your startup before work, not after. Get rid of you TV set. Block out huge blocks of time on weekends. Use your PTO for your startup. Work from home a few days a week and squeeze in extra startup work with the time/energy you save. You get the idea.

You're already creative enough to build a startup. Now use that creativity to free up more time and energy to work on it. Forget about the competition and time slipping away; just do the best you can. And don't pull a Teddy Roosevelt.

143. Should I learn or build first?

"My goals with programming is to create web and desktop (mac, iphone) apps if that's relevant."

Then you have it backwards. You should be building, not reading.

Slow way: Read book --> apply what you learn

Fast way: Write code --> Get stuck --> Find a book

I know this is not intuitive, but trust me, it works much better. We all love the feeling of cracking open a fresh new book (or pdf) and bathing ourselves in all this newfound knowledge. But this method is not very efficient. Much of what you read you will never use and much of what you need you will never read about, no matter what the book is.

Better to pick a project and just start building it. Come up for air every once in a while and consult whatever book fills in what you need to know to build your project. True learning comes from building, not reading. This method takes the best of both worlds and gets you to your stated goal much quicker.

144. What's the big deal with startups?

"What's the big deal with startups anyways..."

1. I write software: business applications. I love what I do. I love getting something to work right the first time. I love seeing people use the software I wrote to do their jobs and run their businesses. I can't imagine doing anything else.

2. The software I have inherited in all 88 companies I have worked at has sucked. I mean really sucked. Nothing to be proud of. Nothing to want to work on. I think it's because business software is now where medicine was 100 years ago.

So I have a choice. Work on other people's crap or write my own. I have done both, but I have to write my own to be happy in this industry. If I could only work on other people's software, I think I'd rather work in a grocery store.

Starting a software business is the most direct way to do what I "really" want to do.

I realize that other people have different reasons; this is just one answer to your question.

145. How do you find inspiration?

"Tell me about how you launched a billion dollar company from your apartment with stolen office chairs and I'm there. Tell me how you really like pointers, and I sort of lose interest."

Thank you. I thought I was the only one.

95% of the time I program. 5% I conduct business.

But for learning, the ratio is reversed. Whether it's hacker news, the articles I read on the web, or the books on my shelves, my interest is mainly in business stories, "especially" start-up success stories.

Not really sure why. Maybe because I think I have all the technology I need. If I need more, I'll find it and learn it. Always have, always will. It's nice to learn a new technique here and there, especially with data base and web technology, but that's rare.

The business success stories, on the other hand, almost always fascinate me. I love Founders at Work and get inspiration from those who have accomplished so much. If regular people like them can do it, then so can the rest of us.

146. How to Never Say "No"

I never say "No".

I just say, "Yes. And this is what it will cost you to do it right:"

```
- Projects X, Y, Z will all be pushed back 2 weeks.
- Prerequisite Project  will have to come first.
-  weeks overtime for  people = $z.
- Joe and Mary will have to be pulled away for 3 weeks.
- Interim solution  will only take 1 week, but  won't work.
- We will need your top supervisor full-time next week.
   or, best of all:
- We don't know.  We need a project to find out.
```

Note that "doing it wrong" or "doing it quick & dirty" are not options.

People understand "this is what it will take" a lot better than "no". They also understand the trade-offs and sacrifices needed. Then they will work with you to make the best decision for everybody.

147. Why I'm a Late Bloomer

My father taught me to read when I was 2 and from that point on, "everyone" encouraged my parents to "fast track" me. I was tested, examined, and prodded by psychologists, doctors, teachers, and "experts". I even passed the preschool entrance exam before my older brother (he's been paying me back ever since).

Finally one day, my father, of all people, said enough. I would mainstream with all the other kids because he didn't want me to be a "freak". To this day, I don't know if that was a wise decision or a snap judgement.

So I sat in class, bored to tears for the next 12 grueling years. Looking back, I had no choice but to "let my love of something pull me". So I learned a musical instrument, started several small businesses, made home movies, and published my own magazine. I excelled in everything outside of school and did poorly in class. I wonder what college admissions officers thought about a self-published C student with perfect SAT scores. I think my magazaine did more for my future that anything from school.

After a great college experience, I spent years of torture in corporate cubicles, bored to death no matter what the job was. Only when I found a way to do my own software startup, did everything fall into place for me. I'm finally living the life I was always meant to live.

So this would-be prodigy ended up being a late bloomer. I don't know whether this is better or worse, but I sure am glad I finally ended up where I belong.

148. How does one turn out the way they do?

The old town drunk died. His two sons, the bank president and the new town drunk were at his funeral. An onlooker, surprised at how different the two sons were, asked each one how he turned out the way he did.

The bank president responded, "With a father like that, how else could I turn out?"

The new town drunk responded, "With a father like that, how else could I turn out?"

For what it's worth, I am like the bank president. I have no idea why. All I know is that no matter whatever anyone ever did to me, it didn't matter. I have no idea if someone who turned out like the new town drunk can change (although I imagine it happens all the time). All I do know is that "it is possible" for a victim to succeed and overcome all of his "darkness".

149. How important to society is your software?

"Have you found a way to write software that has a real, tangible, positive net effect on society?"

Yes. Everything I do.

I (along with many others) write lots of software to help small and midsize businesses compete more effectively. They, in turn, provide value to their customers, jobs for their employees, and pay taxes to their communities. If that's not a "real, tangible, positive net effect on society", I don't know what is.

Not everyone has to find a cure for disease or discover how to provide clean air and water for the masses. You don't have to change the whole world, just a little piece of it.

A mentor once suggested I think of it like a football team. Some pass, some catch, some run, some block, and some tackle. All you have to do is "your job" well, for the team "the rest of us" to succeed.

Please don't be such an "idealist" that you never find your calling. Just pick something close enough and go for it.

The real heroes are in the pits every day, helping others do their thing. You oughta join them.

150. What is "Intellectual Horsepower"?

I used to be awfully hasty in judging others, "She is really smart," or "He is so stupid". Then I learned alot from my first mentor. He taught that there often isn't much difference between someone who appears smart and someone who doesn't. Perhaps no one spent enough time with them. Maybe they have other challenges, like family, health, or circumstances. Maybe they're just a fish out of water, spending too much time on things that don't interest him. Or maybe they appear dumb because they actually believe that they are. They've been told so many times that they now believe it.

At first, he sounded like some hippie idealist. But the more we worked together, the more his teachings manifested themselves in the people we worked with. People who appeared dumb blossomed under different circumstances all the time. The were smart deep down inside where no one ever explored. (These people were mostly hourly workers who knew way more than their bosses about running the business.)

To this day, when I see phrases like, "intellectual horsepower", I cringe. We "smarties" aren't that much smarter than most other people, if we are at all.

And just to stay humble, remember: we're all just one head injury from blissful ignorance.

151. The Disconnect Between Us and Them

I don't know about the rest of the world, but lots of us sure are in a bubble. There seems to be a real disconnect between what people want to build/invest in and what people in the real world actually need and want to pay for. Just as sample of what I've witnessed in the past few years:

```
Ask HN: How do you like my file sharing app?
Ask HN: How do you like my social app for niche ?
Ask HN: How do you like my twitter app?
Ask HN: How do you like my facebook app?
Ask HN: How do you like my iphone app?
Ask HN: How do you like my facebook app that writes twitter apps?
Ask HN: How do you like my game?
Ask HN: How do you like my photo sharing app?
Ask HN: How do you like my video sharing app?
Ask HN: How do I monetize my free flashcard app?
Ask HN: How do you like my app that helps other hackers to do ?
Ask HN: How do I get traffic to my freemium app?
Ask HN: How do I get angels/VCs interested?
Ask HN: Look what I wrote this weekend!
Ask HN: Look what I wrote in one night!
Ask HN: Look what I wrote in 7 seconds!
```

```
Customer 1: How can we sell through Amazon.com?
Customer 2: How can we reduce inventory by $300 million?
Customer 3: How can we increase conversion from 2% to 4%?
Customer 4: How can we use software to reduce energy costs?
Customer 5: How can we migrate one app into another?
Customer 6: How can we get our phones to talk to our legacy apps?
Customer 7: How can we take orders through the internet?
Customer 8: How can we get our software package to do ?
Customer 9: How can we reduce credit card fraud?
Customer 10: How can we increase SEO effectiveness?
```

Customer 11: How can we connect fulfillment and ecommerce?
Customer 12: How can we increase revenue?
Customers 13-200: How can we increase profitability?

152. Weakness or Strength?

"Inability to absorb too many details verbally" = personal fortitude to insist that others show a modium of discipline and occasionally write down what they want

"Inability to multi-task" = ability to focus

"Inability to manage or even to see certain classes of mundane details" = ability to distinguish the difference between and "issue" and a "detail"

"Inability to organize" = lack of the need to organize because of intense focus on the most important thing

"Capable of working through entire books of information" because of the ability to distinguish between "issues" and "details" (see above)

"Capable of coming up with pretty good project ideas on his own" = creativity

"unusual degree of empathy" = understands the "big picture"

"Faults tend not to show up terribly badly" but only to those tuned in to the "superficial", not the "important"

"If he can team up with..." = understands synergy

"Is this guy sick?" = a one eyed man in the land of the blind

153. Why use a framework?

Here's the dirty little secret that no one wants to talk about...

The purpose of "assisters" like frameworks and higher level languages is NOT to make good progammers more efficient.

It's to make mediocre programmers more likely to produce "something" of value and to make poor programmers capable of producing anything at all.

And if the bell curve tells us anything at all, it's that these "tools" target 90% of all programmers.

But think about it, my fellow top 10%, do you really need all this stuff? If you're working alone or on a small team with a clear objective, haven't you always had everything you needed with low level tools? If you need any higher level tools or reuseable components, haven't you already been building these all along?

Sure it's fun to play with new things and learn from others, but when it comes time to really produce, don't we all know how to (and need to) roll with what we know?

The need for frameworks and high level languages only becomes apparent when we grow so large that we can't find enough senior hackers. Only when you dip down into the mediocre masses do you need this help.

154. The Things That Go Without Saying

"1. OO code is less performant than procedural code"

Never forget, the primary purpose of OO is to help "us", not our users. OO is a great way to get junior people thinking a certain way, set standards, and make maintainability a little more manageable (usually). The only thing it really does for our users and customers is help us help them by making our lives a little easier.

"2. The backend is the most important part of development"

Sometimes I think we get this backward. Think about it. You can do almost anything you want on the back end and no one notices unless there's a problem. The client side is a whole different story. It has to work perfectly no matter what browser or resolution your user arrives with, and it has to do it using a limited number of technologies, a small footprint, and with limited round trips to the server. It's almost like it's 1965 all over again.

"3. Graphical designers are good at user interface design"

UI design != UI function. It doesn't matter how pretty it is if the user can't figure out what to do or if it doesn't "function" as expected.

"4. The existence of a superior programming language"

Yea, some languages are better than others for certain things, but a expert practitioner of a seemingly inferior technology is almost always better than an average practitioner of a superior technology. To this very day, my mother can still "print" faster than any of us kids can "write". Amazing.

"5. XML is more economic than a DB"

XML serves its purpose of being autodocumenting quite well. That's all it does well. If you need to parse for any

reason, almost any other format blows it away.

155. How do you feel about competition?

One thing bothers me about this whole WorkHard / WorkSmart / BeProductive meme: it focuses too much on "the competition".

I know my approach is heresy in some parts, but bear with me...

I understand that there's always "some" potential competition, but I choose to not pay much attention to it. The only thing I compete with is another version of myself in another universe. "What would that other Ed have done?"

And that's awfully hard to measure. I've tried all kinds of metrics to keep my projects going, but the only one I use now is how much progress I make each day on my most important task. Pretty subjective.

I have seen tons of good software and services and have done a lot of work deploying them. What invariably happens is that there is no solution for something the consumer wants, so that's what I write. I like to think, "If I had good competition, I would just go out and sell it for them. But since I'm writing something no one else has, I won't worry about competing with anyone but what I would have been."

This may sound a little silly, but it works for me. Provide something that no one else is providing and working harder or smarter than the competition suddenly doesn't mean so much. I just have to be a whole lot smarter and better than doing nothing at all.

156. Levels of Pissedoffedness

Level 0: You don't know that anything is wrong. You just think that's just the way it is.

Level 1: You know something is wrong, but you don't know what to do about it, so you just go along with the program.

Level 2: You know what to do about it, but aren't yet able to do it. So you stick it out, learning as much as you can.

Level 3: You know what to do about it and you are capable of doing it. Now you're really pissed off (mainly at yourself) because you're a fish out of water, in a place where you don't belong.

Level 4: You do something about it. You challenge the people at work to fix things. You start fixing them yourself. Or, best yet, you just go out and do it right on your own. Either way, sweet relief.

Get to Level 4. The days of pissedoffedness will soon seem like a distant memory.

(I have been through this cycle many times, but now I'm at Level 4 and have no intention of ever going back.)

157. My Favorite Business Quotes

Attitude determines outcome. - Jim McGraw, COO of Marion Laboratories

Be the first, be the best, or be different. - Jacyln Easton

Chance favors the prepared mind. - Louis Pasteur

Elegance is for tailors. Don't always believe in the numbers. There is always room for human judgment. - Albert Einstein

Great ideas come into the world as gently as doves. - Albert Camus

Half the money I spend on advertising is wasted, and the trouble is, I don't know which half. - John Wanamaker

I do not love the money. What I love is the making of it. - Philip Armour

I guess we can make them, although we never have. - Benjamin Franklin Goodrich

I never gamble. - J. P. Morgan

A man to carry on a successful business must have imagination. He must see things as in a vision, a dream of the whole thing. - Charles Schwab

If I could get $25,000, I would spend $24,000 on advertising, the remainder in making Coca-Cola. Then we would all be rich. - John Pemberton

If I had six hours to chop down a tree, I'd spend the first four sharpening the axe. - Abraham Lincoln

The best way to really enter minds that hate complexity and confusion is to oversimplify your message. The lesson here is not to try to tell your entire story. Just focus on one powerful differentiating idea and drive it into the mind. That sudden hunch, that creative leap of the mind that "sees" in a flash how to solve a problem in a simple way, is something quite different from general intelligence. If there's any trick to finding that simple set of words, it's one of being ruthless about how you edit the story you want to tell. Anything that others could claim just as well as you can, eliminate. Anything that requires a complex analysis to prove, forget. Anything that doesn't fit with your customers' perceptions, avoid. - Jack Trout

If you're going to lose money, lose it. But don't let 'em nose you out. - Gustavus Swift

If you love an idea, that is good. If you have ideas as to how to work it out, that is better. - Henry Ford

It's a barrier to entry because you're shooting at a moving target. - Bill George, CEO, Medtronics

Name the greatest inventors. Accident. - Mark Twain

Nothing, not all the armies of the world, can stop an idea whose time has come. - Victor Hugo

Perceived truth is more powerful than truth itself. - Michel Fortin

Purchasers are made, not born. - Henry Ford

Success depends on how you react to unexpected opportunities. - Ross Perot

The ancestor to every action is a thought. - Ralph Waldo Emerson

The march of improvement in any given field is always marked by periods of inactivity and then by sudden bursts of energy which revolutionize existing methods sometimes in a day. - George Eastman

There was never a great character who did not sometimes smash the routine regulations and make new ones for

himself. - Andrew Carnegie

Think a lot. Say little. Write nothing. - J. P. Morgan

If first an idea is not absurd, it has no hope for survival. - Albert Einstein

To lead people, walk behind them. - Sun Tzu

We study the methods of improving our business as we would a science. We imitate no one. - A. Montgomery Ward

We took what was a luxury and made it into a necessity. Our only advantage was lack of precedent. - Henry Ford

We want character to go with our goods. And 16 ounces is a Swift pound. - Louis Swift

What others could not or would not do we would attempt, and this was a rule of business which was strictly adhered to. - Andrew Carnegie

What we believe is based upon our perceptions. What we perceive depends upon what we look for. What we look for depends on what we think. What we think depends on what we perceive. What we perceive determines what we take to be true. What we take to be true is our reality. - Gary Zukav

You can't get wet from the word "water". - Alan Watts

You can't solve a problem with the same mind that created it. - Albert Einstein

Chapter 7

Building Stuff

158. How do you collect requirements?

Better to ask first.

Problem is that systems analysis is a lost art. People don't know HOW to ask.

People know what they want. They just don't know that they know.

Here are just a few to the things you have to do (that almost no one does anymore) in order to elicit requirements:

- Spend time with users doing what they do, asking questions until you understand how things work. If this means sitting at their desk, following them around their office/factory/route, or wading hip deep in , then DO IT.

- Ask questions to groups of people at the same time, so they can argue with each other and learn together what's really needed.

- Ask lots of "Yes/No" questions.

- Ask lots of "On a scale of 1 to 10..." questions.

- Ask lots of "What is the probability that ..." questions.

- Cover the wall with paper and draw pictures of EVERYTHING. Leave the paper there for all to see for weeks. Let them understand and learn together.

- Put every piece of paper they use in their jobs (or personal life) on the wall, so they (and others) can really see what they have to deal with on a daily basis. Not blank sheets, but paper with real data on it.

- Identify every data element. How do you know when you're done? Keep asking until there are no more answers.

- Feed people while you ask them. (Amazing how smart people get while eating pizza.)

- Then (and only then) give them a prototype to play with. (I suppose here is where your "measure" comes in to play.)

- Exactly right? No? Keep going until it is.

Aside from the prototype, we haven't written a single line of code. Is this hard? Absolutely. Maybe that's why most of us don't do it anymore. Does it work? It you know what you're doing: every time.

(Aside: Notice I never mentioned using crap like Rational Rose or UML. That stuff was never meant to help this process. It was meant to make people who don't know what they're doing appear as it they did. Built an industry by paying junior people $50k and billing them out at $250/hour.)

159. How can clever software help customers?

Just a few off the top of my head:

1. As part of the research for requirements for a new inventory package, I noticed that every pallet was counted by 3 different people and the lowest count was recorded. I worked with plant supervisors to fix the procedures. Management then realized that there was now no need for new million dollar software. They rewarded my effort and concern for the company with lots of great project work and money. Lesson: Look for the obvious first.

2. A user asked me to help solve her forecasting problem. The two of us sat down and designed the software to do it. I realized there was a parallel effort to do the same thing in another division (with an expensive purchased package), so I made my software work for both divisions. It took 3 weeks to write and people were very grateful. I was employee of the month and got a nice bonus. Lesson: Sometimes little things can solve big problems.

3. I noticed that warehouse pickers were bending and climbing ladders a lot, so I suggested modifying our inventory system to place the most popular items in bins between the knees and shoulders. The change took one week and made us 10% more efficient (a lot of money after a few months). I would have never thought of it if I hadn't been walking around, trying to understand how my software was being used. Lesson: Give yourself the chance to find opportunities.

"Did you build anything that you later spun off into a better job or a side business?"

Yes. Everything I learned using these methods went into 2 businesses: a small business software package and a consulting practice. If I hadn't stretched myself, who knows what cubicle I'd be sitting in today.

160. What's a minimalist coding style?

"A minimalist lifestyle does not make you a better person"

But a minimalist coding style "does" make you a better programmer.

I really don't mind a few extra Philips screwdrivers, kitchen knives, or pairs of shoes in my house, but I every superfluous bit of code in my repository drives me nuts.

Others say I go overboard and they're probably right, but I can't help myself.

If a 6 character variable name can be shortened to 5 characters without losing meaning, then I do it. Same thing with labels and function names. If I find the same line of code twice, I write a function (but only after whipping myself). Complex If Statements are replaced by Case. Complex Case Statements are replaced by arrays and pointers. Two programs look alike? Replace them with one parameter-driven program. Two forms look alike? Replace them with a flexible form app. Reports? Same thing.

Old data? Archived! Old programs? Archived! Old notes? Archived! And not one trip to Goodwill, just to my e: drive. I'm so proud of myself when I can fit the software needed to run a $100 million company on a 256K thumb drive.

There must be a 12 step program for people like me. But then, by the time I was done with it, it would be a 7 step program.

161. How do you build someting piece by piece?

I love how everything has a fancy name now. We've been doing StartInTheMiddle / MinimumViableProduct / Prototyping / GetSomethingOut / StepwiseRefinement for years:

Customer: I must have anything I want from the database without asking you.

Me, 1 hour later: No problem, here's your screen.

Customer: This only does does Customers. I want to pick "any" table.

Me, 1 hour later: No problem, now you can pick your table.

Customer: This dumps every column. I want to pick my own.

Me, 1 hour later: No problem. Now you can pick your columns.

Customer: This doesn't sort. I want it to sort.

Me, 1 hour later: No problem. Now you can sort.

Customer: But I want multiple sorts, some ascending, some descending.

Me, 1 hour later: No problem. Sort any way you want.

Customer: It dumps the whole table. I want to filter.

Me, 1 hour later: No problem. Now you can filter.

Customer: It only give me local columns. I want columns from other tables, too.

Me, 5 mins later: Give me an example.

Customer: Here. Give me these linked columns and accumulators, too.

Me, 2 days later: OK. I figured out how to give you all this data too.

Customer: OK. This will work. Why didn't you do all this in the first place?

Stepwise Refinement Method: Time to beta: 1 hour. Time to production: 3 days.

Waterfall Method: Time to beta: not applicable. Time to production: who knows?

162. What does a programmer/analyst do?

Long gone are the days when the "system analyst" met with the users, wrote tight functional specifications and handed them to the "business programmer". In any organization that actually gets anything done, the two are now one position, the "programmer analyst", and it's been that way for 20 years now.

And what does a programmer analyst have to do?

- examine and understand almost any business situation

- solve technical and logistical problems

- present ideas clearly and cleanly

- organize and manage time and tasks

- play nicely with others

- and, oh yea, code

And where do you find people who can do these things? Lots of places, but the 2 that come to mind first are from work experience or college experience. Nobody actually believes anymore that you'll use anything from class in your work. But it can be a very good place to develop the life skills needed to be a programmer analyst.

Nobody's going to ask you to write a linear programming model using the simplex method to determine how to allocate continuous inventory to customer orders. (OK, maybe they will.) But they will wonder why Mary and Joe can't seem to figure out how to use Program ORP560 to generate this month's performance metrics. Is the problem with them or the program? They'll tell you to figure it out and fix it. Count on it. And count on college to help you become the kind of person who can do that.

163. Programming FAQ

What editor do you use?

Textpad.

How can I learn to program?

Find a customer with unreasonable deadlines. Hit them. Repeat. It won't be pretty, but you'll be the kind of programmer I'd go into battle with: great at the things that really matter and mediocre at the things that don't.

Why do you advise plunging right into a programming project instead of carefully planning it first?

It's incredibly difficult for anyone to define what they want when they start with nothing. It's human nature (incredibly easy) to criticize what you have and how to make it better. Plunging right in and producing anything raises you from LevelIncrediblyDifficlut to LevelHumanNature.

Why do you keep going on about PICK?

I have a very small back pocket that holds everything you need to program in PICK. Whatever you can do in technology , I can probably do in PICK, often quicker and easier.

Isn't object-oriented programming naturally suited to some problems?

Sure, but that doesn't necessarily mean you have to use formal object-oriented technology. You can mimic the concepts with almost any technology.

164. Are these things really that hard to do?

"I have not seen these things be successful, therefore they "can't" be successful".

This kind of thinking scares me, not so much because it's so myoptic, but because it becomes a self-fulfilling prophecy: "I have never seen success" becomes "Success is not possible" which becomes true because we stop trying.

Mere mortals throw up their hands in futility, blaming the user, the state of the art, or the alignment of the planets, without ever understanding the fundamental principal that great systems accomodate great needs and the greatest need of all is constant change.

"the company recategorizes its product line based on utility and lifestyle rather than brand and designer"

This is incredibly common and can easily be accomodated with proper database design, decoupling products from their associated attributes.

"the parent company divests itself of you and withdraws the connection to its parts and inventory database"

So now you have to treat them like any other vendor? You do have some of those, don't you?

"a manufacturer starts using more than one UPC per product because they changed the packaging"

A simple database design consideration. "One to one" is just a special case of "one to many".

"the law now requires that national holidays must be counted against vacation time if the employee takes a vacation day immediately before/after that holiday"

Rule driven logic. What's the big deal? (Your logic was rule-driven from the outset, not hard-coded, right?)

"a loan company files to become a bank and store deposits that earn interest"

Additional logic to existing data. We do this all the time.

"CSV files exchanged with a partner were always denormalized, but then one company starts using normalized records with mixed column mappings"

Any decent batch file processor should be able to handle anything thrown at it. Companies that do a lot of this build uncertainty into the design of their systems.

"another partner decides to ditch file transfer as a way to submit data, and creates a web service that you're supposed to invoke commands on"

Sounds like you'll need an extra piece of software to create a feed. Not exactly like no one's ever done this before.

"the shipping carrier reveals that their tracking numbers are all recycled at the end of each year"

Not a problem if they were never used as the primary key to a table.

"new rules require every credit card number to be encrypted, but you still have to be able to quickly search for matches in a database of millions"

Ahh, but the last 4 digits do not have to be encrypted. Start your search with them and resolve synonyms. Pretty standard stuff.

"a partner switches to mixed-case alphanumeric PO numbers"

As an identifier, a PO number is always a string even if it looks like it's numeric. (When was the last time you added 2 PO numbers together?)

OP's roadblocks are experienced developers' speedbumps. Just because he hasn't done it doesn't really mean that "Nobody knows what they're doing".

165. Is becoming a lazy programmer evolving?

Show me any tool and I'll show you a horrible use of that tool. That doesn't make the tool horrible.

ELSE statements don't kill. Drunk programmers kill.

This reminds me of something I once ran into:

Junior Programmer:

```
if (m==1){Month="January")}
if (m==2){Month="February")}
if (m==3){Month="March")}
if (m==4){Month="April")}
if (m==5){Month="May")}
if (m==6){Month="June")}
if (m==7){Month="July")}
if (m==8){Month="August")}
if (m==9){Month="September")}
if (m==10){Month="October")}
if (m==11){Month="November")}
if (m==12){Month="December")}
```

Senior Programmer:

```
switch(m)
{
case 1:
  Month = "January"
  break;
```

```
case 2:
  Month = "February"
  break;
case 3:
  Month = "March"
  break;
case 4:
  Month = "April"
  break;
case 5:
  Month = "May"
  break;
case 6:
  Month = "June"
  break;
case 7:
  Month = "July"
  break;
case 8:
  Month = "August"
  break;
case 9:
  Month = "September"
  break;
case 10:
  Month = "October"
  break;
case 11:
  Month = "November"
  break;
case 12:
  Month = "December"
  break;
default:
  Month = "unknown"
}
```

Lazy Programmer:

```
MonthNames == ["","January","February","March",...]
Month       = MonthNames[m]
```

166. How are we making this too complicated?

I have had people come to me with seemingly complex problems requiring what they thought would be complex solutions. Oddly, the elegant solution was often a gross simplification of the complex problem. Often because I just didn't understand the complexity.

Example 1. A shipper can only fit 1200 boxes in a 40 foot trailer. The material is so light, they can never make weight. The boxes must be delivered to depots throughout the United States with no more than a 3 day window either way. What box should go on which truck? The customer thought they needed sophisticated algorithms to provide an optimal solution. The simple solution: a screen where an educated user can drag boxes (or groups of boxes) to trailers and can drag trailers to destinations. The computer did very little, but the difficult business problem turned into a "game" that employees fought each other to "play".

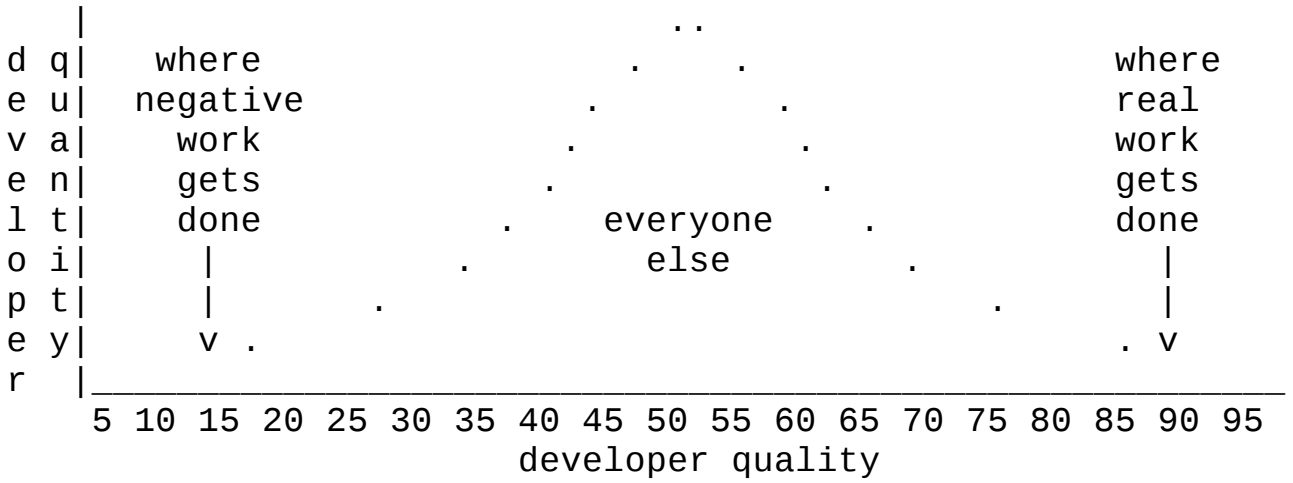
Example 2: A cloth manufacturer wanted a linear algebra solution to determine what products to mount on their mills and where to set the knives. The simple solution: A screen with all customer orders and key data about each order (size, weight, width). Everything on the screen was sortable and switchable. Again, an intelligent user could "play a game" to schedule the mills almost as well as any automated algorithm.

Example 3: A distributor wanted to know what to buy, when to buy it, and what to put on sale to move slow inventory. He wanted an expensive ERP program. The simple solution: A screen with all supplies and demands that can be mixed, matched, sorted, and selected in many different ways. Again, an intelligent user can "play" with his data and then confidently make decisions.

The "real" bonus of all these "simple" solutions? The person drove the decision and felt in control of the situation. No more blaming the computer for the results.

My suggestion: Slow down. Even stop. Ask, "How are we making this too complicated? What simple little thing could I do to get just part of the way there?" You may surprise yourself and find that the simple little idea IS the solution.

167. How Stuff Gets Done



168. Can waterfall planning work?

If you come to the realization that work in itself isn't evil, you can stop living your life as a waterfall-planned software project too.

Nothing wrong with waterfall-planning if it's done properly. Problem is, it usually isn't.

I understand that sometimes you need to release early and often. This is when you can't conduct proper anaysis. Why not? Because you're trying for a home run building something big and you don't know where your project will take you or who will eventually use it. Like a Web 2.0 or social site.

But for the rest of us, waterfall-planning is just another name for the Systems Development Life Cycle (SDLC), which is an excellent way to develop systems. But you must conduct analysis first. You must answer the question what before you examine the question how.

Most developers do not know how to do this. How can you tell? When their waterfall phases take too long (more than a month for analysis for most projects). When they say things like, The user doesn't know what he needs. Yes he does. You just have to keep digging until you know.

Once a good programmer learns how to conduct analysis he becomes a good developer. Then projects become like shooting fish in a barrel. You biggest problem will be convincing your customers that you can do what they think can't be done.

169. What makes a programmer senior?

Great question. Ask it to n programmers and get n^2 responses. This could easily be the subject for another post or even a book. Just off the top of my head in no particular order:

```
- understands the problem at hand before writing any code
- uses the right tool for the right job
- follows accepted standards and protocols without sacrificing creativity
- names variables & functions what they actually are for the next programmer
- anticipates what could go wrong before relying on a debugger or testing
- understands the underlying architecture and how to best utilitze it
- never writes the same code twice
- never writes in 150 lines that which could be written in 100 lines
- Poor code: uncommented.  Mediocre: commented.  Good: doesn't need comments.
- understands the entire code life cycle & writes it to last
- has pity on the poor soul who has to maintain it & leaves a clue or 2
- writes flexibly enough to be easily changed before the project is done
```

I could go on and on, but you get the idea. In general...

A good programmer writes it right, once, in a week.

A mediocre programmer writes it OK, in 2 months, and then futzes with it forever.

A bad programmer never gets it done.

170. What are relational databases so important?

Today's action items:

- Give me a list of good (lifetime sales > $10,000) customers on the west coast (CA, OR, WA) who bought any product on the defective list during busy season (10/06, 11/06, 12/06) and haven't placed an order since our last email blast. We'll find out why.

- Give me a list of phone numbers (using our Caller ID) of people who called in the last 4 days, whose call wasn't answered and had never called us before. We'll call them back.

- Give me a list of our slowest moving (10%) products that haven't been ordered in the last 4 months that are sitting in prime space (Location beginning with 1,2, or 3) in the warehouse. We'll move them to pallets to make room for new stuff.

- Give me a list of all products returned from Territory 7 in the last 3 months with a Problem Code related to fit or size. We want to make sure their description is correct on the website.

I could go on (and on and on)...

Just because something is old doesn't mean it's obsolete. Maybe because it works so well.

171. Is software engineering dead?

"Software Engineering is Dead" is obviously an overly sensational title, but let's look for the deeper truths.

First a little background. I have built a career developing business applications as a contractor, often in very large organizations. I am almost always very successful and I'm often regarded as a hero doing what most people here would call "just doing my job".

Why is it so easy to be a hero in the enterprise with software that would just seem ordinary in other places? Is it because enterprise programmers aren't as good as us? Is it because their managers are all phbs? Or because they don't know how to run projects?

I'd say no to all of the above. Enterprises are filled with lots of excellent people doing great work.

I think that the real problem is that the Systems Development Life Cycle (SDLC) that so many depend on so much "never really did work". Why?

Every phase depends upon Phase I, Analysis to be rigorously done. This rarely happens for 2 reasons: users often don't know what they want and most systems analysts don't know how to extract it even if they did.

So almost everything done after Phase I is built upon a foundation of sand. It's either wrong or sinking fast. And what do most people do? Everything except fixing the problem: more resources, more project management, freezing specs (which aren't right in the first place), more rigorous deadlines, etc.

But rarely does anyone attack the core problem with the Systems Development Life Cycle: defining the expected result.

So what should we really do? Develop something, anything, quickly, cheaply, and get it out to the right users. They will instantly give you feedback. What's right, what's wrong, what's stupid, all the cool stuff that no one thought of.

No one can just sit down and write a Functional Specification for a large business application. And even if they could, you don't want them spending time on it. Better to get the right people together and find out what they need. Usually, no one of them knows what the result should be, but all together, any decent developer should be able to extract enough data to write version 1.0 of "something".

It's a lot easier to judge something that exists than define something that doesn't.

The larger the organization, the more difficult it is to change their ways.

Software engineering isn't dead. It's just that the process of depending upon blueprints before you get started never worked in the first place.

172. Why is BASIC still OK?

"It is practically impossible to teach good programming to students that have had a prior exposure to BASIC: as potential programmers they are mentally mutilated beyond hope of regeneration."

For what it's worth, I have written over 1 million lines of BASIC for over 100 customers, most of it still it production, and all of it doing important work producing goods, services, and jobs in so many uncool industries we couldn't live without.

Maybe I'm an outlier, but I have gone on to learn algorithms, dynamic programming, database theory, client/server, and web development. I believe the elegant simplicity of BASIC and database theory, although limited in application, has provided an excellent base upon which to build.

I know that ewd is a giant to be respected, but I think it's a red flag when a teacher mutters "practically impossible to teach", even in jest. IMHO, that says more about the teacher than the student.

Thoughts like this are great for a laugh, but when you stop to think about it, all they really do is further amplify the perception of a huge gulf between theory and practice. Academics whine while those of us in the trenches are too busy to notice because our sleeves are rolled up while we build that which must be built.

173. When should you rewrite?

When it's a house of cards.

Perhaps I'm a little jaded, but I've built a very nice career rewriting software that never should have been written in the first place. Of the existing code I've encountered, at least 95% would never have passed a code review by me (or anyone else who knew what to look for).

The problem is that we're so committed to pass user acceptance testing that we never subject the source code and data base design to the same rigors.

I like to think I've seen it all: one and two character variable names that mean nothing, homemade routines for , memory leaks, iterations to nowhere, upward branches to nowhere, data base schemas that would make M. C. Escher jealous, and on and on and on. Face it, if programmers were doctors, we'd all be dead.

As a constant victim of the "You Can't Get There From Here Syndrome", I often tell my clients the same thing, "It's not how soon we get started, it's how soon we finish AND how capable we are to handle the next revision".

Often rewriting is the last best hope.

174. How can you clean input data?

If you're going to "diagnose the state of your data", why not clean it up. There's so much that you can do at entry time, at retrieval time, and at any time between:

```
- wash non-printable characters
- wash illogical (depending upon context) characters
- trim leading and trailing spaces
- verify check digits
- verify lookups
- verify against standards (USPS, etc.)
- add Soundex, Metaphone, levenshtein, etc.
- build a context suitable hash
- add Soundex, Metaphone, levenshtein, etc. to the hash
- wash standard keywords
```

175. 10 Signs You're a Crappy Programmer

10. The exact same code is in multiple places because you didn't bother to put in into a common function.

9. You have error codes in functions, but never bother to look at them (a sure sign that testing was never finished).

8. You have early exits from loops because you don't know how to properly code a recursion.

7. You execute too much code because you don't know the difference between "if" and "case".

6. You use 10 lines of code when 1 line would do because you don't know any better.

5. You need to run batch jobs from time to time to clean up your data base because you don't know how to debug well enough to find what's messing it up.

4. A third grader doesn't know what your variable names mean.

3. You think "Third Normal Form" is paperwork you need to fill out for one of your 8 bosses.

2. No programmer who has to maintain your code would ever want to work with you.

1. Any programmer that has to maintain your code would be more than happy to accept the consequences of first degree murder.

176. Technical Debt

Typical way that competent developers end up with technical debt:

1. Customer/user/consumer has a vague idea of what they need but doesn't know how to express it.

2. Developer works with customer to learn their business and understand their requirements.

3. Developer throws together a prototype to confirm that he understands the problem.

4. Customer sees that developer is on the right track. If these 9 enhancements are done, then we'll really have something.

5. Developer quickly adds 9 enhancements to prototype.

6. Customer loves it! Move it to production so we can play with it for a while.

7. While customer plays with it, developer maps out a plan to architect, refactor, and scale the prototype to be "production worthy".

8. Developer is pulled away to 5 other urgent projects.

9. Customer continues to use prototype as production software.

10. Two years later: "Who wrote this crap?"

177. Annoyances vs. Requirements

"One of the annoyances that we have to deal when building enterprise applications is the requirement that no data shall be lost."

Since when is a business requirement an "annoyance"? We keep data for lots of good reasons. Just a few off the top of my head:

```
- IRS requirements
- SEC requirements
- SOX requirements
- data warehousing
- data auditing
- delayed undo
- business intelligence
- trend reporting & analysis
- research & development
- cooperative databases
- industry databases & statistics
```

"The usual response to that is to introduce a WasDeleted or an IsActive column in the database and implement deletes as an update that would set that flag."

Wrong. The usual response is archiving to disk using separate tables. This solves all of the requirements above while streamlining active database tables. It's not unusual for 98% of all the enterprise data on disk to be in the archived state.

"This sort of cleanup now has to moved up into the application layer."

So what?

One of the biggest mistakes I consistently see is trying to use the facilities of a database management system to replace application logic. Indexes, triggers, and stored procedures are all critical tools in the developer's arsenal, but they do not replace application analysis, design, and programming. Like security and scaling, archiving is a "architectural consideration", not a bolt on. Including proper data archiving in the application's design renders the hard vs. soft delete debate pointless.

178. What can be optimized?

I've always thought there were 2 types of things that could be optimized:

1. Things that need to be "cleaned up".

2. Things that never should have been written in the first place.

Simple example of Type 1: You rush to get something up and running, and in your first code review, you find the exact same code multiple times. So you write a function, parameterize a few variables, tighten it up, and reference it all over the place. Cool.

Simple example of Type 2: You have an SQL SELECT inside an iteration. At 500 iterations it runs smoothly. At 50,000 iterations, it becomes non-functional. Your only hope to scale this thing is to rethink the whole process to run with one SQL SELECT (and maybe a database redesign) outside the iteration. You basically have to start over. What were you thinking?

You need to trust your "process" that Type 1 things will rise to the surface in due time, thus avoiding premature optimization.

For Type 2 things, there is no such thing as "premature optimization". They need to be designed and written properly in the first place.

179. Why pre-develop?

"I've never seen anyone able to design something away from keyboard that doesn't change significantly once it's written"

It "does" change once it's written.

The idea is to get a clear work plan on a "close enough" design. I estimate that my first cut of anything is maybe 50% or so.

The idea is also to avoid sitting at the computer all day and then being disappointed with how little I accomplished. Activity != accomplishment.

A little more background...

First term freshman year, 90% of science students took Chemistry I. On Mondays and Wednesdays, only 50% of the seats in the dining room were taken for dinner. Chem Lab started at 1:00 p.m. and dinner was at 6:00 p.m. So, most freshman chemistry students took more than 5 hours to complete their lab work.

This never made sense to me. I took Chemistry I second term freshman year. My lab partner and I made a pact to "never" miss dinner. We did everything we possibility could to expedite lab time. We did all the reading, planning, and reviewing other people's results "before" we entered the lab. We even wrote our reports in advance, filling in the results as we went. Our longest lab took 2 1/2 hours. Our shortest took 1 1/4 hour. (We also both got A+.)

I still practice that methodology today. My computer is my lab and my bed or sofa is my lab prep. Preparation takes as long as it needs. Labs go fast. If they don't it's because I wasn't prepared enough when I started.

180. Documentation Belongs in the Code

The advice from this post is exactly what you'd expect in theory and exactly "what not to do" in practice. For one simple reason: the source code is (hopefully) the only thing pretty much guaranteed to survive.

I have seen countless shops where valuable history was lost because it was stored on someone's c: drive, a network drive, or some repository that failed to survive some kind of migration. And even if these other files (digital or paper) did survive, chances are that the programmer that needed to see them never did anyway.

Good shops practice keeping audit trails "in the source code". This means good commenting. Which means good code review and quality control.

I recently came across a single piece of code that had been changed back and forth 6 times in the previous 2 years. The comments looked something like:

```
* jeo 02/11/09 Use Ship Date, not Book Date per Sarah in Sales
* jrm 04/15/09 Use Book Date to make military contracts balance
* msl 08/24/09 Use Ship Date per Joe in Ops (military no longer active)
* jrm 12/13/09 Use Book Date per Rick Smith to prepare for new contracts
* jrm 02/14/10 Use Ship Date per Rick Smith after Ops meeting
* jrm 05/25/10 Use Book Date per Rick Smith until Q3 migration
```

I know that this is an extreme example, but this stuff happens all the time in commercial environments. How easy do you think it would be for the programmer/analyst to provide background if these comments were not in the source code, but somewhere else?

Sure it's a pain it the ass to maintain this, but it immediately provides the needed background to the person who needs it, when he needs, where he's already working. For critical projects with confused users (what isn't), the

alternative is usually much more work.

181. What's the best way to assign ID's?

The "Name" issue isn't that much different from the "SKU" issue. (SKU is short for Stock Keeping Unit, aka Part Number or Product Number). I have had to deal with this everywhere that has SKUs. No one does it well, but by slowing down and thinking about it, there's almost always a decent solution.

There are 2 ways to assign SKUs, sequentially (start with "1" and increment 1 for every new SKU) or not sequentially. I have never seen anyone do it sequentially. (Although some excellent systems keep 2 SKUs, one of which "is" sequential and is used as the primary key and for all indexing. This is the best way I've ever seen to handle SKUs that change, but that's another story.)

Almost everyone wants a smart or semi-smart SKU. So that by simply looking at the SKU, anyone can tell what it is without reading the description. You know, the first digit is Commodity Code, 1 for shoes, 2 for pants, etc. Then another digit for color, another for size, etc. This works well until you have ten colors; then you need 2 digits or alphas.

But wait, there's more. Let's put hyphens (or some other delimitter) between the product descriptors and the vendor data, manufacturer data, and customer data.

So now you've covered any possible product with your super slick smart SKU naming system.

Until something comes along that isn't covered. (Now we have military items with 14 other considerations.) So we come up we a second totally different scheme. Then a third. Then a 4th, etc. So now you can tell anything about an item "if you know which scheme" it falls under.

But wait, there's more. You should be able to enter any SKU into a form field "regardless of Smart SKU scheme". (If the first digit is "9", then use Smart Scheme 3. If there's a hyphen in position 3, then use Smart Scheme 8, etc.) Your form logic should be able to intelligently guide the user based on the rules of the template or scheme.

I have built apps where users can design and build their own Smart SKU templates, which are then used to enforce compliance and guide operators. These have generally worked pretty well.

Is there some way to do the same thing for human names? I dunno, but now you got me to thinking about it. A combination of standard templates and custom templates oughta cover most possibilities. Some basic logic with optional pop-up forms which uses the templates as parameters should work. Something to think about...

182. Why do you hate bad design so much?

Bad design is "everywhere"...

My old microwave oven had one control, a dial. Just put the food in and turn it. The whole dial represented 12 minutes, so just estimate how far to turn it. My new microwave has 22 buttons. I forget which is which. I have to turn on the lights and get my glasses.

The previous owner installed vertical blinds on the double hung windows. Think about that. Impossible to have open windows without endless noise and movement.

An office where I work is near an airport. They have a key card system that beeps with a successful swipe. So you can't hear the beep when an airplane is passing overhead (every 5 minutes) and have to wait until it's gone to get into the building.

Another office has "gone green" and installed motion sensors everywhere to turn off lights not being used. Good luck finishing up in the restroom if you've been sitting too long.

Some of my software asks "Save before exit?" "whether I've changed anything or not". I got so tired of trying to remember if I had updated or only viewed, I just click "Yes" every time. Pointless.

Every time the garbage truck drops the dumpster back down next door, 3 car alarms go off. No one ever responds, because they assume that it's a false alarm. Not that big a deal until the garbage truck shows up at 4:00 a.m. on Saturday.

My TV remote has 33 buttons. So they're so small, you have to stop and look at it to hit "Mute" or "Last Channel", the only 2 buttons I ever use. Who designed this thing, a munchkin?

The airports in Orlando, Philadelphia, San Jose, and Miami put the restaurants (not counting junk food) "outside" of

security. So you have to wonder, "Do I have enough time to eat and then get through the line?" (Pittsburgh, Tampa, and Chicago Midway did it right.)

The trunk release and gas cap release levers in my car are next to each other but not visible. It's hard to pull one without the other. Seems like my gas cap door is always open.

A theater we went to the other night only had aisles on the sides with 50 seats in between. Do your really want that great seat in the center knowing you may have to climb over 25 people if you need the restroom before the movie is over?

Intersections that gridlock because of traffic from the next light. Too many to mention.

Microsoft Windows.

183. Why do you hate old code so much?

Single entry/single exit refers to ANY process, not just functions. Don't underestimate the hell who can go through with poor variable naming. Case in point:

2 months ago, a client needed significant changes to a major process in their app. (I had 2 weeks.) This process was a BASIC subroutine called by 16 other processes. Now get this: It was 2800 lines of code with one entry and 16 different exits. It was written in 1991 and had been modified 68 times before I came along. Here are a few of the variables: A, AA, AAA, AAAA, B, BB, BBB, BBBB, C, CC, CCC, CCCC, INFO, FORM1, FORM2, FORM3, HOLD.FORM2, OLD.FORM2, ORIG.FORM2, STUFF, DUMMY1, DUMMY2, DUMMY3... You get the idea. I spent 4 days renaming variables before I could start refactoring. There were some variables I never did figure out. I rewrote the program (550 lines) and hit the deadline. I uncovered over 20 undocumented bugs. I wonder what it will look like 5 years from now.

Everything I write new is either Javascript or PHP. Now that my startup is ramping up, soon I won't have to deal with any of this old stuff any more.

(Aside: I wonder how much production code in the world today is over 10 years old. And how much has been modified over 10 times?)

184. What makes code crappy?

When it comes to maintaining code, I believe there are 2 kinds of crap: subjective (I don't like it) and objective (it's crap because of these 14 specific reasons).

You're probably right about people who inherit my code. I know, when they've whined about it, I confronted them. "Please show me exactly what's wrong with it. What are your specific complaints about violations?" I rarely got an objective answer. It was usually something about formatting, indenting, variable names too long, variable names too short, I'm use to it the way we did it (wrongly) at XYZ Co., something like that.

True crap can be objectively identified by a violation such as:

- variables named so that no one except King Tut could possibly figure out what they are

- the same variables used for different purpose "at the same time", usually in nested recursions

- variables named with 1 or 2 characters

- unassigned variables

- variables initiated when they shouldn't be

- division by zero

- single entry, multiple exit (heavily maintained so that now outlying cases skip critical logic)

- the same code in multiple places (only some of it maintained so that outlying cases skip critical logic)

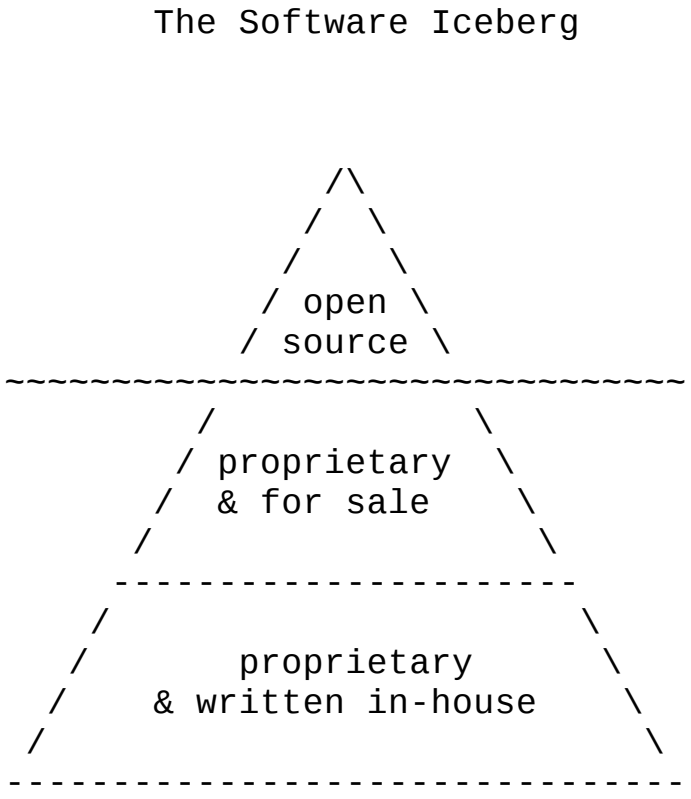
- endless recursions (for outlying cases only, naturally)

- comments that don't agree with the code

- data base tables with no definitions either in the schema or any code (my new favorite)

I could go on and on, but you kinda get the picture. I wonder how many readers here have posted crap like this on their "wall of shame" at one time or another. It's funny the first 2^n times. Not so much fun anymore.

185. How much software is open source?



186. How far should automation go?

"Weak human + machine + better process was superior to a strong computer alone and, more remarkably, superior to a strong human + machine + inferior process."

I had to read this statement 3 times before it hit me: What's true in chess is also often true in business. A little background...

I recently wrote a forecasting system for a company that processes 7 million orders per year. Worse, this company was the merger of two other companies, each of which did forecasting differently. One had a very expensive Oracle based "strong comptuer" that calculated almost everything and told the planners exactly what to do. The other just dumped data into Excel files and teams of "strong humans" manipulated them until they intuitively worked out the best plan. Neither team could believe the way the other team worked.

The system I wrote using guidance from both teams turned out to be "weak human + machine + better process" which leveraged the strengths and minimized the weaknesses of the two extremes.

Chapter 8

Software Business

187. Why start your own business?

One day you realize that you only have x days on this earth and y of them are already gone.

You don't want to waste any more of your remaining (x-y) days refactoring the same poorly written crap for the eighth time, answering the same 84 inane emails, drinking the same lousy coffee, looking at your watch through another pointless walk-through meeting, and listening to the idiotic pontifications of a boss who you would never talk to in a million years if you weren't here.

You know you can do better. You know you have it in you. You know you can make a difference. Then you know you "have" to. So you get out your calculator, work your finances, and when you have it all figured out, you turn in your notice and enjoy the best day of your life.

188. I'm sooo confused...

```
Release early and often.
Get eyeballs, then monetize.
Bootstrap.
Get basic version working, then add features.
Get basic version working, then scale.
Get basic version working, then license the technology.
Get basic version working, then sell to enterprises.
Get free version working, then upgrade them to premium.
Get free version working, then sell support.
Get free version working, then sell services.
Get free version working, then put it on your resume.
Raise money, hire wisely.
Be first to market.
Be second to market.
Wait for market to clear, then enter with second wave.
Find your niche.
Have the lowest price.
Have the highest price.
Use technology as a barrier to competition.
Win a business plan competition.
Get into an incubator.
Get into a seed accelerator.
Find an angel.
Find a mentor.
Crowd source fund raising.
Get press to raise money.
Raise money to get press.
```

I think I'll just fall back on the only thing I know:

```
Build something people will pay for.
```

189. The Investor Entrepreneur Chasm

Investor: What are you building?

Entrepreneur: Artificially intelligent software that automatically builds sophisticated business applications based on the enterprise's business rules.

Investor: Your competitors are too entrenched. What can you do that's simpler?

Entrepreneur: Small business software that ties all a company's applications together.

Investor: You'll never compete with Microsoft. What else?

Entrepreneur: Tiny apps that all kinds of people can use to run their stuff.

Investor: 37signals will kill you. What else?

Entrepreneur: Social software that enables your sales people to understand what's happening in the global marketplace.

Investor: It'll never work. Can you do something more practical?

Entrepreneur: An intelligent e-commerce system that guarantees the consumer the best value.

Investor: You'll never compete with Amazon or Ebay. Got any other ideas?

Entrepreneur: Recipe software.

Investor: OK, if that's the best you can do, we'll go with it. Geez, I just wish you guys would dream a little bigger.

190. How can I get started?

1. A small business has problems and knows that there must be a solution, but doesn't know what's possible or where to turn.

2. You make contact. Through a personal introduction by a friend, relative, or business associate. Or at an industry event (their industry, not yours). Or at a chamber of commerce event. Or any local business event. Or by mailing them a postcard, flyer, or letter using a purchased list or phone book. Or in a restaurant, bar, or party. Or from a flyer or business card that someone else gave them. Or from an ad you ran in their trade publication. Or from a search that landed them on your website. Or... (you get the idea, it could be anything).

3. You meet and listen. I cannot stress this enough. This is 100% about their problem, not your solution or anything else.

4. If you have a way of addressing their problem, do it. It may be software you've already written, a service, or (quite likely) a prototype you mock up to show them how to attact their problem.

5. They love the fact that someone has finally actually listened the them about their problem and addressed it. Your solution is a good first step, but it still needs a, b, and c.

6. You quickly add a, b, and c to your prototype and show them.

7. They're in love. Now you can get started.

191. What went wrong in your first start-up?

In my first failed start-up, I did what is now considered standard advice and it was an utter failure (which might explain why I still question all advice, no matter how standard).

The software was a small business system for manufacturers and distributors. I was the technical person, my co-founder was the business person.

How we did things:

- We determined the customers' most critical requirements.
- We built what they needed from those requirements.
- We installed the hardware and software.
- We got them up and running in test mode.
- We adjusted, reworked, and went live.

What ended up happening:

- Critical features were invariably missed. I had to add them.
- There was always some scaling issue we missed. Always.
- Architecture had to be reworked with every install.
- My co-founder was able to sell far faster than I could build.
- My co-founder was unable to help me build.
- Customers became disillusioned.
- I collapsed, vowing never to go through this again.

What I now believe:

- Make sure your MVP is enough.

- Beware being consumed by customer service.
- The first two founders must be technical.
- Your architecture must scale, even if your app doesn't.
- Always be brutally honest with each other at all times.
- Make sure all your failures are recoverable ones.
- Plan for 40 hours/week. Stop working at 80.
- Never quit. Start over, but never quit.

192. Why are details so important?

I remember the time I took 4 enterprise vice presidents from New Jersey to visit a software vendor in Silicon Valley. They had great (multi-million dollar) software and it was perfect for this customer.

Our first meeting was at 9 a.m., and there was no coffee! The Vice President of Sales actually found the coffee and filters and brewed the first pot himself in the vendor's conference room. (Probably the first time he made coffee in 20 years.) Then he said, "Why should I trust them to handle my customer orders when they can't even do the basics right?"

A subtle but very important point. When engineering people sell to business people, we have the extra burden of showing that we know how to conduct business at their level. The easiest way to get started is with precise attention to details. And faux pas destroy trust much quicker with web technology.

193. Why are you writing your own software?

Because the guy who wrote this:

```
if (MonthNbr == 1) {MonthDesc = "January"} ;

if (MonthNbr == 2) {MonthDesc = "February"} ;

if (MonthNbr == 3) {MonthDesc = "March"} ;

if (MonthNbr == 4) {MonthDesc = "April"} ;

if (MonthNbr == 5) {MonthDesc = "May"} ;

if (MonthNbr == 6) {MonthDesc = "June"} ;

if (MonthNbr == 7) {MonthDesc = "July"} ;

if (MonthNbr == 8) {MonthDesc = "August"} ;

if (MonthNbr == 9) {MonthDesc = "September"} ;

if (MonthNbr == 10) {MonthDesc = "October"} ;

if (MonthNbr == 11) {MonthDesc = "November"} ;

if (MonthNbr == 12) {MonthDesc = "December"} ;
```

is a retired millionaire while I'm still cleaning up his mess.

I think I can do better.

194. When do you say "yes"?

"So here it is, the most plain, powerful, single word you have to know, and use when managing a project: NO"

If you aspire to mediocracy in an enterprise, then this "may" help you survive. Otherwise, it's horrible advice.

If you have serious competitors, then you have to find YES.

If you are attempting to do something extraordinary or for the first time ever, then you have to find YES.

If you are building a startup, then you most certainly have to find YES.

I am not saying that all things are possible. I am saying that you need to find YES. Once you get into the habit of saying NO, you forget how to find YES.

A simple (and timeless) example. Your customer wants Deliverable X in Time Y using Resource Z. You know it's too much and will disappoint. So instead of saying NO as OP recommends, you find a way to do what can be done. It may have a few less features, may need an extra resource, or may take a little more time.

Or better yet, you analyze the constraints long enough to find methods or tools you hadn't considered to say YES to all of it. (We never would have found Framework ABC if the customer hadn't forced us.)

I have often been to only person finding YES when I was surrounded by others preprogrammed to saying NO. That's how they survived. Usually in an enterprise or institution. That same thinking is a disaster in an achievement oriented environment.

Finding YES forces you to stretch beyond your previously perceived limits. Settling for NO dooms you to mediocracy forever.

Some may call this a semantic argument. I call it state of mind. How badly do you want it? Find YES.

195. What drives development?

I guess I'm a little bit different than others.

I am TOTALLY guided by my customers. I wasn't always this way. I used to think that something would be so cool, so I would build it, and often, the project went nowhere. I was fortunate to have a co-founder at one time who insisted that we sell it first, then develop it. I never completely came around to his way of thinking, but now I understand where he was coming from.

My customers have never steered me wrong. They don't waste my time. They only spend energy describing things that they really need, and invariably, others need the same things.

The downside is that I never spend time working on my own pet projects. I KNOW I can build a better bridge game, fitness program, or home inventory program. I'd also love to blog. But all those things fall into the category of "No one else asked for it", so I simply don't spend time on them. Maybe some other time.

196. Why would you not launch?

0. You're truly not ready.

A recurring meme is the equivalent of "Just Do It". Excellent advice, almost all the time. Almost. Except when it's terrible advice.

Yes, as someone who has suffered after launching too soon, I will go against prevailing wisdom and suggest the unthinkable, "Maybe you're really not ready and can do more harm than good by launching prematurely." Just a few of the bad things that can happen:

1. People will visit once, see that it's crap and never come back again, no matter what you do.

2. You will be overwhelmed by support requirements to the extent that all development stops.

3. You will be overwhelmed by support requirements to the extent that much support never gets addressed.

4. Your calendar becomes science fiction; everything has changed and it's a whole new ballgame.

5. The stress level will become so overwhelming for some of your people that you will simply lose them. Forever.

6. If you have taken people's money and not delivered, the guilt can become so overwhelming that it cripples you.

7. Your marathon has turned into a sprint you cannot finish. You have launched and lost.

I love the idea of pushing the envelope and launching sooner rather than later. You must have real world feedback and launching is best way to get it. But launching too early early is just as bad as launching too late. So how do you know when the time is right? I don't have a definitive answer, but I do know that your gut is a critical input. Sooner or later, you just have to go with it.

197. Does consulting hurt a software start-up?

Consulting does NOT need to be a tradeoff when starting a software startup.

Why not?

If you pick your customers carefully enough, they can be the R & D department for your startup. They don't even realize it and they pay you for the privilege!

You are going to need tons of feedback for your software. One strategy is the well-known "release early and often". Another, just as effective, is "find out from your own customers before you develop". You will still need to release early and often, just not as early and not as often.

I estimate that more than half of the ideas for features in my software came from existing consulting customers. Things I would have never thought of, and now I know they're necessary. Without them I would have been releasing alot more often and early, and may have never received the same valuable input.

"Learn from the work"

I spend less time consulting than many entrepreneurs spend fund raising. I like to think of my customers as "angels whose money I get to keep".

198. What do small business owners care about?

Small business owners are always concerned about revenue. Always. They know there are 2 ways to solve almost any small business problem: (a) Work like hell on 42 different things, or (b) Add sales. Find a way to show them new orders and you will get their attention.

Small business owners often feel like they must defeat someone else in order to win (whether it's true or not). Always leave a little wiggle room in price or terms so that they can enjoy the feeling of a victorious negotiation.

Small business owners appreciate simplicity. Confuse them with technology or jargon and they will move on to something they understand better.

Small business owners understand the importance of other people in their business. Show them that you do too and that will go a long way toward your credibility.

Every small business owner has a few pet peeves that drive them nuts. It might be the outrageous cost of something or how difficult it is to get something done. Discover and solve these things and you could go far.

Small business owners are especially sensitive to bullshit. If you're a poser, save yourself the trouble and move along.

Small business owners often have a bigger "real mission" for being in business. Find out what that is and help them achieve it. They will really appreciate you for that.

199. Product/Market Strategies

Not so Good: Build it and they will come.

Good: Make something people want.

Better: Make something people need.

Even better: Make something people need and know that they need. If they don't already know, someone will have to help them know. That someone might be you and helping them will take sales and marketing, a significant consideration.

Even better: Make something people need, know they need, and are willing to pay for now. Competing against "we're not ready, maybe next year" is tougher that competing against any competitor.

Best: Make something people need, know they need, are willing to pay for now, and has their hair on fire. Leapfrogging to "let's just solve this problem now" is often the best (and most fun) way to deploy.

[EDIT: This is "not" about "guessing what people need", worrying about competition, or assuming "if I need it, others must too". This is about getting up off your butt, talking to people, finding out what they must have, and building it for them. There are people everywhere desperate for solutions to their problems, and I promise you, they're not finding them. Sure, you may scratch your own itch and hope that someone else needs it, but this is a lottery approach. Most start-ups fail because they built something no one else wanted or was willing to pay for.

There is absolutely no better way to find out what to build than finding customers first. Please don't be like me and learn that the hard way. There are great apps everywhere that nobody uses while people suffer because no one is building what they want.]

200. Differentiate or Die

"Their service has more bells and whistles, but mine is much simpler and quicker to use."

You just answered your own question. You must focus your marketing on "simpler and quicker" to the exclusion of everything else. (Either "simpler" or "quicker" would be even better, focusing on "one" thing.)

Jack Trout, in "Differentiate of Die," says it much better than me:

"The best way to really enter minds that hate complexity and confusion is to oversimplify your message. The lesson here is not to try to tell your entire story. Just focus on one powerful differentiating idea and drive it into the mind. That sudden hunch, that creative leap of the mind that "sees" in a flash how to solve a problem in a simple way, is something quite different from general intelligence. If there's any trick to finding that simple set of words, it's one of being ruthless about how you edit the story you want to tell. Anything that others could claim just as well as you can, eliminate. Anything that requires a complex analysis to prove, forget. Anything that doesn't fit with your customers' perceptions, avoid."

201. Are the any advantages for single founders

Top 10 Reasons for Being a Single Founder

10. You spend 0 time debating technical issues that have already been decided.

9. You spend 0 time refereeing personal differences among the other co-founders.

8. You spend 0 time wondering why they can't keep up with you, or why they're doing something other than what you have already agreed upon.

7. You learn every part of your business. You never worry what's happening outside of "your turf".

6. You can always find a sympathetic ear to discuss a technical, marketing, or business issue. For free.

5. You can always find someone else to socialize with, with no impact on your business. For free.

4. You are not going to get hit by a bus.

3. If your significant other wants more of your time, give them a picture of the new house, car, jewelry, or NBA franchise they will own if they exercise some patience. They're the only other one that really matters.

2. You get real good setting up your schedule to work best for you, with "head down" time, and other time.

And the best reason for being a single founder...

1. You keep all of your equity. All of it.

202. How does it feel to be a single founder?

Thank you for the great post on a subject near and dear to many of us.

I am a single founder who constantly wonders if I should be. I have founded 2 startups before and both imploded solely because of founder issues. One had 4 of us and I spent more than half my time playing referee. I will never go through that again. As far as I'm concerned, nothing is more important than being in business with the right people (except having plenty of customers, maybe.)

I am working hard and steady on my new venture, but I do not actively recruit potential co-founders. I freely share what I'm working on (hinting at opportunities for co-founders) and sit back to see what happens. Since I believe the single most important trait of a good co-founder is sheer determination, I hope someone will come to me insisting, "We should do this...," "I could do that...," "Let's try this...", etc. Sorry to say, this strategy hasn't worked too well, but I'd still rather be alone than be with someone who doesn't push as hard as I do.

I know about half a dozen excellent people who I'd love as co-founders, but all have mortgages, families, and other commitments. Our startup would always be a child to me, but only a step-child to them. So I don't push.

I understand the concern about single founder startups; they're a lot of work! There are many times I wish I had someone to share with or help me out. But under the circumstances, I'd rather plow along, always positioning myself to have a co-founder, but just as ready to launch alone if need be. I'm not afraid to do that and I don't think other single co-founders should be either.

203. What should a business guy have to offer?

An overwhelming majority of the time spent on a software startup is at the terminal, coding. If you're not doing that, then you damn well better be bringing something else of value, a lot of value, to the table. Things I'd be looking for...

- Specific domain knowledge. You gotta be the guy who says, "No, no, no, that's not the way you do in this industry. You do it this way. And you sell it this way."

- Our customers' industry contacts. You gotta be opening doors for us while we're busy coding.

- Analysis, design, testing, implementation. You should be very proficient at all the stuff on the Systems Development Life Cycle that's not coding.

- Operations. You should good at running the business while we code. This could include many things like accounting, marketing, selling, order processing, help desk, legal, etc.

We programmers are not dummies. We are working on the critical path, but we recognize that a whole lot of other stuff has to happen in order to be successful. And we want to be successful, which is a primary driver. Can you do that other stuff? Great. If not, you need to find some way to add value.

And don't forget, ideas != value. We all have a million of them. What else ya got?

204. Where can I get help starting a business?

I would ask for help from my current employer.

This, of course, requires you to be completely open and honest about your desires and that they're not jerks.

They already think you're a superstar, so it really won't be that much of a surprise when you say something like, "I had so much fun 'liberating our data' that I'd like to start my own service business doing the same thing for others."

Smart business people like helping others, especially helping others get started. They seem so sense that the karma will eventually come back to them (and I believe they're right). They also understand (even better than you) the business benefits of your work and can really help you focus your new business.

Ask them (starting with the top person, of course) for guidance on how to get started. You may be surprised how much you learn from them and how willing they are to help. You may think you know how much your work has helped them, but I bet they have more to share that you don't already know.

They probably can be the source for great leads, their vendors, their customers, other companies in the boss's CEO or Tech circle, golf buddies, who knows. For example, if your owner/president/CEO has an associate who would benefit from your services, "everyone" wins when he recommends you.

As long as they don't feel their core business threatened by your service business (helping their competitors), your own employer may be the best source for ideas and leads to starting your own service business. Give it a shot.

205. Why should you fire bad customers?

Wanting it fast, good, and cheap is also a red flag for lots of other little bonuses, such as:

- You will constantly wait for them to make a decision.
- It will be your fault they took so long to make a decision.
- They will have emergencies of their own making.
- It will be your fault they have emergencies.
- They will commit to little or nothing on paper.
- It's not their fault because they never committed to that.
- You think you have specs; they think you're prototyping, so...
- You will do much work 2 or 3 times.
- They will constantly change priorities.
- They will forget they changed priorities, so...
- They will complain when a lower priority isn't done.
- You won't get paid on time.
- You will spend lots of time trying to get paid.
- They will always find some excuse to not pay.
- You may never get paid.
- If it's good, it's because they thought of it.
- If it's bad, it's because you suck.
- You can't win.

Honestly, I wish we could tattoo these people to save the next developer all the heartache. As soon as you realize they want it fast, good, and cheap, "run" the other way.

206. How important are ethics?

This subject comes up every month or so, and every time I give my opinion which is always in the minority. Sometimes I think I'm the only one. So once again, as I prepare to get downmodded into oblivion, here goes...

You forgot Side 3 - I hate software piracy because it's wrong. Period. It's unethical, immoral, and illegal. And it's that simple. I don't even consider either of your 2 choices because both sidestep the question of right or wrong to examine other issues. This is situational ethics.

In all the years I've been in business, my number one concern has been ethical issues. The partner who disconnected his speedometer to increase his resale value. The vendor who raised his prices to get a personal kickback. The employee who downloaded a customer list and sold it to a competitor. I could go on and on and on...

I've seen stuff like this so many times, and I ask the same question every time, If they will compromise their ethics on something small, where do they draw the line? I've seen multi-million dollar deals scuttled because someone didn't trust someone else because of their personal behavior on a small issue like this. Don't let yourself fall into this trap. It simply isn't worth it to save a few bucks.

I've heard all the counteraruments. "It's no big deal." "Everyone does it." "It's not hurting anybody." "I'll never get caught." Or the worst one of all, "They've already ripped me off, so I'm just getting them back." And you know and I know and everyone here knows it's all BS. We're just making excuses for what we all "know" is wrong.

Almost every proprietary software vendor has a complimentary "developer version" or a very cheap "student version". There are many other ways to get access to software or music without breaking the law or compromising your ethics. But a lot of us are just too lazy to take advantage of these things.

I would expect programmers, of all people, to be especially sensitive to this issue. After all, we are smart, hard working people who make software. But it seems like I'm always in the minority on this one.

207. Why are ethics so important?

I don't want to debate fine points of ethics, but I thought I'd share a little more background.

I really think that this is a black and white issue. I don't see any difference between illegal downloading and walking out of Walgreen's with a CD in your pocket. Or putting that extra chicken leg from the buffet into your purse. "They'd just have to throw it away, anyway." I don't care. Right is right and wrong is wrong.

I don't ever want anyone to get the impression I'd employ situational ethics in business. And I do not want to knowingly conduct business with anyone that does. It's simply not worth it, period.

I once had a partner that drew the ethical line where it was most convenient for him. First, he copied software from one account to another. Then, he went through a client's employee's drawers looking for something to "save us a lot of time". Before I realized it, he was making back door deals with clients and vendors because he "didn't think I'd mind; it was money I'd wouldn't have ever seen anyway."

I'm not suggesting that everyone progresses down that path, or that reusing tidbits of code is the same as murder. It's just that when it's time to draw an ethical line in the sand, my position is clear and firm.

Just a few antecdotes to give you an idea of how strongly some business people feel about this issue:

- An acquintance of mine was earning $150 per hour advising a Fortune 1000 company which multi-million dollar enterprise package to buy. As an aside, he brought in a buddy to sell printers to his client and split the profit. He was immediately fired and black-balled. The CEO's reasoning was, "I would have never known if we made the right decision."

- A vendor was presenting their software package to my client. They said, "We already know your industry. In fact, we sold a system to XYZ Company." My client immediately dismissed the vendor. He later said, "That's all I need. For one of his programmers to accidently say what I'm doing to an XYZ employee over coffee."

- My client went bankrupt. Their assets (including all IP) were acquired by a third party in the settlement. Imagine their surprise when they had to compete with my client's ex-employee who set himself up in a software maintenance business at 1/2 industry rates. How did he know who to call on and what software they had? The case is still in litigation, but that guy's name will forever be dirt in this town.

- A contractor at one of my clients accidently left a thumb drive on a desk he was using. It had 70,000 social security numbers on it. What were they to think?

I could go on and on. They are some real slime balls out there. There are also plenty of good people who make stupid decisions to save a little time because "it doesn't make much difference anyway". How are people supposed to know the difference?

And when it comes to technology, many business people are doubly in the dark. Sometimes, TRUST is all they've got. It's so ridiculously easy for many of us to earn a nice living (try digging ditches instead), why would you ever jeapordize that over something so trivial?

208. Do you conduct business over meals?

Very interesting topic we don't see much. I'm probably in the minority, but I'd like to share what has worked best for me.

90% of the business I have ever conducted over a meal has been at breakfast. By lunch time, I'm too busy and dinner is usually reserved for family. But breakfast is perfect. You get people when they're fresh and before they're sidetracked. I prefer to have private meetings first thing in the morning and have also regularly gone to Chamber of Commerce, tech groups, vendor presentations, even Toastmaster breakfasts. They're always early enough for most people and work out great if you want to network or sell and still have a day job. And they never run over because everyone has somewhere to go.

And guess what I've eaten at every single one of them?

Nothing.

Business breakfasts are about business, not breakfast. You can do three things with your mouth at breakfast (talk, eat, or both) and two of them are bad. I spend most of my day at my terminal, so the business breakfast is my big chance to talk, listen, and learn. And food just gets in the way.

You can't talk while you're chewing, almost every choice is time bomb for an accident, and in my humble opinion, food just slows you down in the morning.

So I let the others eat while I talk (and listen). I accomplish twice as much as anyone else at these breakfasts.

I just have a cup of coffee that I may or may not drink. (You may look like you're wasting food if you don't eat it all, but nobody cares how much of your coffee you drink). If I'm really hungry in the morning, I'll grab something "before" going to the breakfast, but I'm more likely to wait until afterward.

Jimmy Carter took this idea to the extreme by never eating dinner at dinners. He was too busy networking and conducting business while everyone else was eating. I've never gone that far, but his strategy works perfectly for the business breakfast.

209. What's your favorite start-up book?

Do More Faster by Techstars founders Brad Feld and David Cohen

(Apologies to OP's request for brevity; there's just a lot of good stuff that I'd like to share.)

A must read for anyone here who is serious about their startup.

I read it on the flight to Startup School to "get in the mood". I couldn't put it down.

It's easy to read for 2 reasons: every chapter is a short essay by a different person (including many Techstars alumni) and it's very well written, almost like pg essays but by lots of different people. It covers lots of ground, much of which has been covered here at hn many times, but then again, some of this stuff can't be covered too often. Also, sometimes someone says the same thing a little differently, and that's the one that actually reaches you.

My 300 page copy has 50 or 60 dog-earred pages and hundreds of red marks; it's that full of gems. (For that reason, I highly suggest buying a hard copy and keeping it on your bookshelf for future reference.)

I think that yc should come out with a similar book. I'd love to read essays from yc alumni, their advisors, and of course the yc principals themselves about what they thought was important. I realize much of this is on-line already, but there's nothing like a great hard copy too.

A few of my favorite quotes from _Do More Faster_:

"I realized that I had two options. I could quit buying comics or I could quit my job and build the iTunes of comics." - Kevin Mann

"Getting feedback and new ideas is the lifeblood of any startup. There is no point in living in fear of someone stealing your idea." - Nate Abbott and Natty Zola

"That means every moment you're working on something without it being in the public arena, it's actually dying, deprived of the oxygen of the real world." - Matt Mullenweb

"Focus on the smallest possible problem you could solve that would potentially be useful" - David Cohen

"You know you're on to something when the community starts donating money to make sure it stays alive." - Darren Crystal

"In companies that rely on having a large user base as ours does, it is very unlikely that you will offend enough people quickly enough to dampen your future growth." - Sean Corbett

"We learned that very few people care how you accomplish something. Instead, these people care more about whether you create value for your end user." - Colin Angle

"We knew that the high-level concept of our first site still really inspired us." - Alex White

"They stepped back from what they had created and thought about what they could do better than anyone else in the world." - editors

"During the first few days of every TechStars cycle, we tell the 10 bright-eyed new teams that one of them will not be together at the end of the program. Unfortunately, we have not been wrong yet." - editors

"If you can't quit no matter how hard you try, then you have a chance to succeed." - Laura Fitton

"When you ask CEOs of major companies what they're most worried about, one common answer is 'a couple of guys in a garage somewhere.'" - David Cohen

"Companies that work just always seem to move at lightning pace." - David Cohen

"It turns out that giving up your one obvious competitive advantage often proves to be deadly. If a startup can't do more faster, it usually just gets dead faster." - David Cohen

"There is an enormous difference between exciting technology and an exciting business." - Howard Diamond

"Changes come daily, weekly, and monthly - not once a quarter or once a year." - Ari Newman

"While it was only a detour of a week, that's a lot in TechStars time." - Bill Warner

"Only hope instead is to listen to their head and their heart and follow a path that they believe in, keeping some of the feedback and discarding other thoughts and ideas." - Bill Warner

"...when presented with exponential growth, remember that people tend to drastically overestimate what will happen in the short term, but will profoundly underestimate what happens over longer time spans." - Ryan McIntyre

"...consider life as a founder of a startup to be one big intelligence test." - Ryan McIntyre

"Remember that human nature has a tendency to admire complexity, but to reward simplicity." - Ben Huh

"If you are innovating, you actually don't know what your product needs to be. Furthermore, your customers don't either. No one does." - Ajay Kulkarni and Andy Cheung

"Nearly every startup must find ways to differentiate itself from competitors." - Raj Aggarwai

"What is the thing that matters most to making progress right now?" - Dick Costolo

"...you cannot create the need." - Michael Zeissner

"Opportunity cost can kill a startup." - Michael Zeissner

"It's easy to feel trapped by these handcuffs but if you change your perspective just a little, you might find that you hands are bound by nothing more than air, and the future is yours to create." - Eric Marcoullier

"There is one thing that the hundred of founders I meet each year have in common, and that is that their plan is wrong. Sometimes it's the big things, sometimes it's the little things, but the plan is always wrong." - Rob Hayes

"...we have to strike while the iron is hot! My experience is that this is rarely true." - David Brown

"Take the time to get it right and you'll find that those competitors might not be as close as you think." - David Brown

"Seeking the perfect combo: 'a smart-ass team with a kick-ass product in a big-ass market.'" - Jeff Clavier

"The moral of the story is easy: When you follow your heart, good things usually happen. We have a very short stay on this spinning orb and I believe life is way too short to be stuck in a career that doesn't fulfill you." - Mark Solon

Chapter 9

Enterprise Life

210. Willie Sutton would be an Enterprise Programmer

Enterprise software sucks.

We don't talk about it much, but think about it. Every man-made object you encounter every day was manufactured somewhere. And moved, more than once. Now add in all the sales, marketing, customer service, operations, accounting, finance, human resources, etc., needed to support that manufacturing and distribution. Next, add financial markets, healthcare, energy, entertainment, etc., and you have tons of stuff. But you don't see it and rarely think about it. Kinda like most of the iceberg being underwater.

And all of this needs software. And most of what they have sucks. I mean really sucks. Enterprise software is so bad that there are multi-billion dollar industries devoted to consulting on how to use it, how to share it, and how to store it in data warehouses and harvest it. It's so bad that lots of people have to dump the data out of their enterprise systems and into Microsoft Excel just to get anything done.

When Willie Sutton was asked why he robbed banks, he said because that's where the money is.

What banks were in the 1930's, enterprise IT is in the 21st century.

211. What do enterprise people need to know?

Trivial example of what people need to know in enterprises:

"Tell me whether pet rocks are selling better than Barbie dolls in the south?"

What people really need to know:

I already know from existing reporting that 984 orders (18% of our backlog) are already past due. For those 984 orders:

```
- How many are for one item and how many are for multiples?
- Do we own what we owe those customers?
- If we do own it, is it in the proper warehouse?
- If it is in the proper warehouse, can we find it?
- If we can find it, is it undamaged and certified?
- If it's shippable, do we have enough labor to ship it?
- If it isn't certified, how soon can QA certify it?
- If it isn't in the right warehouse, can we move it?
- If we don't own any, where can we get some?
- Which vendors have it on the shelf?
- Which vendors do we have blanket purchase orders with?
- Which vendors do we have contracts with?
- Which orders can be split to satisfy a partial?
- Which orders are for customers already on credit hold?
- Which customers are threatening not to renew with us?

and (ironically) the most asked question of all:

- Which orders must be shipped to hit our quarterly numbers?
```

I can go on and on; this is just off the top of my head. We like to pick on enterprises, but this is the stuff that happens all the time. So whenever you get gas in your car, bread on your table, new shoes at the mall, steamed milk in your latte, etc., rest assured that "someone", "somewhere" has asked these questions. Questions that were probably answered using some form of RDBMS, SQL, ACID technology (with really good application software on top of it).

212. What is a typical enterprise day like?

Yesterday I was up to my earlobes implementing a business intelligence/data warehouse system in a very large SOX-compliant enterprise :-).

For current requirements, this solution is excellent. It is third party software, installed on top of an existing ERP system, that enables the users to extract whatever data they need and build their own reports without submitting a ticket to IT. Everyone loves the prototypes and is dreaming about the possibilities.

So instead of beating up on enterprise life, let me just share a little bit of yesterday, one typical enterprise day:

- A 3rd party build just stopped at 98% complete with no error message.

- Another build crashed with error messages I had never seen, so I had to open another ticket with our vendor

- We ran out of disk space on a volume we didn't know the software was using.

- I had to add additional data cleaning functions to remove heretofore unknown control characters in the enterprise data.

- I inadvertantly named 44 files with the vendor's own naming convention, so now no one can tell whose files are whose. We had to reset our standards and rebuild.

- Although this vendor has hundreds of installs, oddly, none of them are SOX compliant. The controls, audits, and duplication of data needed will more than double the resource requirements. Worse, I'll have to do an implementation that no one has ever done before with this software :-)

- Today we start writing our own tools to handle the SOX compliance and satisfy the auditors. Some fun.

I can go on and on. You get the idea. And I haven't even touched upon the usual enterprise culprits: the meetings, the politics, and the lack of project management. It kinda sucks to have to do triple work to get the same thing done.

So whose fault is all of this? No one's. That's just the way it is. Every time I think of a better way to get things done in a large enterprise, someone has 5 good reasons why they are the way they are. It's wasted energy fighting that.

213. What's is enterprise IT's biggest fear?

In enterprise IT, we have 2 kinds of vendors, those we actively embrace and those who hold us hostage.

The biggest fear in making any major IT purchase is not the price, the conversion, or the change in culture; it's the potential loss of options in how we run our own business.

I've seen it over and over again: competitive pressure requires us to make a change in the way we run our business, but we can't. For all kinds of reasons. The license agreement kills any possible ROI. We don't have the needed IT support because so much of it is spent on keeping current. The systems don't talk to each other. The feature we need is still 18 months away. And probably most of all, the software is not as excellent as we need it to be (let's just leave it at that).

Long gone are the days when you needed IBM's permission to fart. Guess who the biggest culprit is today?

Just because you go with someone doesn't necessarily mean you like it. Almost every enterprise IT department I know would love an alternative to Microsoft. (And make no mistake about it, the enterprise is Microsoft's strength, much more so than the consumer.)

Sure, pissing off your customers may pad today's bottom line. How do you think those customers will feel when your landscape changes and they have more choices?

[Entered using ie7 on xp pro. I didn't have a choice.]

214. Why do excellent programmers leave enterprises?

There's an elephant in the room: your best people leave because you're not paying them enough.

And you're not paying them enough because you've chosen to adopt a pay structure that, by definition, cannot handle outliers (which are exactly what your best people are).

An excellent programmer can routinely do the work of ten mediocre programmers. Have you ever heard of a pay scale that pays Programmer III ten times as much as Programmer I?

An excellent programmer can save you millions of dollars by increasing revenues, reducing costs, changing the way you do business, or any combination of these. How much of those millions does the programmer get?

An excellent programmer can provide a company significantly more value than his boss and his boss's boss. How much more does he get paid?

An excellent programmer can earn real equity and merit bonuses in enlightened firms. How many companies don't even have this on their HR radar?

An excellent programmer can go out and pound the pavement or join a start-up and earn what he/she is really worth. And until most companies provide that same opportunities, they probably will.

215. Where are there real problems to solve?

"Aren't There Real Problems To Solve?"

I have always spent a lot of my time in small or midsize companies (10 to 200 employees), and I can assure you the answer is ABSOLUTELY.

These are hard working people who are in a different world than most of us. They don't facebook, twitter, or IM at work. But they are unbelieveably busy trying to get things done while depending upon software and systems that most of us here would laugh at.

Many of them don't have time to get to all their emails and voice mails. They have to make snap decisions all day long without enough information because their software just doesn't give them what they need easily enough. They have constant problems with interpersonal communication, instruction, and understanding.

Just a sample of what I've witnessed in the past 3 weeks:

- We need multiple "ship to" addresses for this institutional customer. The software doesn't support it.

- The exchange server is down.

- We need multiple prices for the same item, depending on the scenario. The shopping cart won't allow it.

- 300 "smart part numbers" were configured incorrectly because no one gave Jane the correct Vendor Codes last week. But now, we have orders, inventory, and history for those items, so the software won't let us change or delete them.

- The exchange server is down.

- No one remembered to reset the warehouse server last night, so everything came in today with the wrong date.

- The new stockroom printer doesn't support HP PCL, so the bar code labels are all wrong.

- We think that $90,000 order acknowledgement from XYZ Company was deleted by the spam filter, but we're not sure.

- There was a bad pointer in the UPS data base last Tuesday, so 72 packages all went to the same customer in Duluth.

- The wrong 800 number is being printed on our invoices. Our customers are calling a sex hotline.

- The exchange server is down.

These are not jokes. I see them all the time. And while we regale all the cool stuff WE are doing, I can honestly say that the situation in many small businesses hasn't improved all that much. And there are 7 million of them. They need help.

The good news? This is a fantastic opportunity for those with our skills and technology.

Yes, there are plenty of real problems to solve. Stay tuned.

216. How should layoffs be handled?

1. Help each laid off employee land on their feet, whatever that means for them. This must NOT be lip service, but a legitimate effort. Hire an outsourcing firm, provide resume/career counseling, provide reference letters, or find job opportunities with vendors, customers, or industry contacts.

2. Make a clean transition. Give laid off people an opportunity to share what they were working on and debrief others. Even though they are leaving, many people value the work they have done and want to know that it's in good hands. If you don't do this, you might as well be saying, "We can figure out what you were working on without you." Nothing else you say or do will offset the damage this will do.

3. Find a way whenever possible for people to keep contributing as contractors or part-time employees.

4. Find a way for people to keep their health insurance.

5. Offer severance whenever possible.

As uncomfortable as a layoff can be, it's also an opportunity to show how well you can do the tough stuff. People will be watching and remembering; count on it.

217. How do you test drive packaged software?

I just converted a client OFF of a major enterprise package to something 20 years OLDER and much easier to use. Frankly, I was stunned to see how difficult it was to use and how poorly it was designed.

Examples: Even though it maintains 160 columns for each Order Header, "Time Entered" is not one of them. So go ahead and try to schedule a call center with no Time Of Day history. To review a single customer's history (and presumably answer their questions while they're on the phone), you have to open up 6 different windows!

My general suggestion: Write a "Test Drive" based explicitly upon your client's business. Have them enter an order and take it through every stage of it's life (reserve or commit inventory, print, pick, ship, confirm, bill, collect, post, etc.) The test drive should also hit every function needed to do this, setup a customer, SKU, GL Chart of Accounts, etc. Then have the appropriate person working for your client do their job on that system. DO NOT touch the mouse or keyboard. DO NOT let the vendor touch the mouse or keyboard.

If they can't enter, ship, and bill an order in a couple of hours with less than one day's instruction, eliminate the system and save yourself a lot of expense and headaches down the road.

Naturally, most sales people will object to this approach and use any tactic to avoid it. Make it clear to them and your client that this must be the plan. The test drive method is infinitely more effective than the old RFP approach. When the poor systems are clearly failing the test drive, the saleman may object with questions like, "Are you going to train your people or not?" which you'll calmly answer, "Not if it takes until 2018." Good luck.

218. Have you ever been a hero at work?

Many times. A few of my favorites:

- We upgraded our hardware and our forecasting software vendor wanted a one time $250,000 charge. I convinced my boss to replace them with in-house written software. Took 6 weeks to write.

- Our 400 worker factory was $30,000 under-absorbed per month. I wrote both standard costing and data collection software. Supervisors compared the standards to the actuals to discover where they were losing money. We were over-absorbed by $30,000 per month 6 months later.

- We budgeted over $1 million for a new ERP system to "solve all of our problems". I helped others solve most of their problems by identifying them and coming up with solutions from the existing software. We never did buy new software.

- (My favorite). Our HCFA feed from the U.S. Government was broken and no one knew why. I dug in and changed 1 byte of code (1 byte, not 1 line). The next day, our bank account had $6.5 million more in it. I never had the heart to tell them how easy it was to fix.

219. Why is SQL so important in enterprises?

Try designing an enterprise production/distribution system where:

```
- set-ups must be minimized
- backlog must be minimized
- inventory must be minimized
- trucks must be full
- warehouse space is limited
- deliveries must be on time, but not too early
- sales people must have stock on hand
- plant absorption must be maximized
- bills must be paid on time
- down time must be zero
- working capital must be put to the best use
- stockholders must be satisfied
```

Say what you want about enterprise programmers, but they get stuff built that handles all of these while academicians screw around with linear algebra and OO castles for years.

220. How did ERP get so screwed up?

ERP has deep roots.

The original acronym was MRP, Material Requirements Planning, a perfect candidate for business software. It answered the question, "If I need to deliver 9 helicopters on these 9 dates, then what components will I need on which dates?" Believe it or not, this was all hand calculated at one time.

MRP was very complex and difficult to implement because it required absolute precision and discipline, rare back then and still rare today. If your base data (inventory balances, lead times, quantities per, etc.) were the least bit off, the resulting automated explosions would be way off. So an industry of software vendors and consultants was born to attack all of these issues.

The problem with MRP was that it didn't work well at all for products with few components but complex processes, (think chemicals, energy, distilleries, food processors, etc.) So CRP, Capacity Requirements Planning was born to plan and manage factories with high capital expenditure requirements. (It doesn't matter if we have exactly the components we need if we have nowhere to work on them.)

Before you know it, "everyone" wanted in on the act of expensive software and consulting, even in disciplines that didn't require them (why should SAP make all the profits). So along came accounting, sales, HR, and everyone else, and now we're stuck with ERP, a cow that's ripe to be milked for a long time.

221. Why is ERP becoming a dinosaur?

I remember leading the test drive of a popular ERP system for one of my enterprise clients. The salesman had never had a prospect force him into a "test drive". My number one rule was "Nobody touches the keyboard except me."

As I entered an order, I asked, "How do I find my Customer Number?"

As I entered a part number, I asked, "How do I see how many are in stock?"

As I scheduled the order, I asked, "How do I see the current factory schedule for this item?"

It drove the salesman nuts. He couldn't explain how to do anything without grabbing the keyboard.

I asked, "What good is this system if I can't even enter one order?"

His response..."Are you going to train your people or not?"

That's pretty much their attitude. He was so upset with me, he went directly to the CEO, who asked me what was wrong. I handed him one of our orders and said, "If you can enter this order into that system, we oughta buy it." He couldn't. We ended up buying another system that people could use.

The dinosaur is big, but it is dying. Hack on, and let the one who delivers the value win.

222. How tough is on-line retailing?

I know lots of online retailing millionaires. Is it easy? No. Is competition fierce? Of course. Are margins slim? Usually. But if you have passion for an idea, a market, a good plan, and are willing to work your butt off, then there are fantastic opportunities.

(Aside: You won't get a whole lot of response from people making their fortunes in on-line retailing in December - many of them are working 20 hour days this month.)

I have often thought about starting something, but I'd rather hack. (I think that anyone can retail, but few can hack - I'm not so sure how empowering this thinking is.)

Things you'd better consider before taking the plunge:

- How will you find customers? Search engines, email blasts, catalogs, list buying, renting, or sharing, advertising...

- How will you fulfill orders? Your own garage, rented space, dropshipping from your vendors,...

- What capital will you require and where will you get it? (This may be the biggest barrier vs. hacking)

- How much volume will you have to do to break even? (Better be right about this or you're dead in a hurry)

- What people will you require and how will you manage them?

- What systems will you require and where will you get them? (Needless to say, there is great opportunity for competitive advantage here.)

- What makes buying from you so much better than anyone else? (Maybe the most important question.)

- There are hundreds of other things to consider, but you get the idea...

People have started with a small idea are parlayed it into hundreds of thousands (and even millions) of dollars per year in less than a few years. The case studies are almost endless. Google a few and see how they did it. Or watch Donny Deutch's show "The Big Idea" on cable TV.

If you decide to do something like this (not the best time of year now since everyone will be tapped in a few weeks), find your niche and go for it. Best wishes.

223. Why would we want to go faster?

I remember a project I worked on in a large enterprise years ago. All of their systems, believe it or not, were batch. Inventory, accounting, order processing - all data was entered into hold files, or worse, filled out with pencil and paper and turned into keypunch. All databases were updated in a large batch overnight. (Today, it's hard to believe anyone ever did that.)

Our project was to migrate all apps to a new real time package. They spent millions of dollars and when it was all done, the controller complained, "Who decided that we needed real time Accounts Payable? Why would we ever want to pay our bills "faster"?"

No one had ever asked that question before. No one even thought about it. We had spent $1 million on a module nobody wanted because IT decided it. Eventually he added procedures to continue to fill out all accounts payable transactions with pencil and paper and enter them into the on-line system at the end of the day. What a waste.

Same argument here. Some things you want faster. And you're willing to pay for them, one way or the other. But other things should just stay the way the are.

Some things never change: you actually have to "think" about your apps before you implement them.

224. When can code review be a problem?

A customer recently asked me to look into a program that used to run in 5 minutes but now took 1 to 4 hours. It's used by thousands of people all over the world all day long.

It iterated through an array doing 3 SQL SELECTs against non-indexed files for each element. There used to be about 50 elements in the array; now there were more than 5000. I rewrote the whole thing in one day to do a total of 4 SELECTs and run in 12 seconds.

But it took 6 days to get through QA (while the users continued to suffer). QA's biggest complaint? I indented 4 spaces instead of the (unpublished) standard of 5 spaces.

Of all the things I have to deal with, nothing pisses me off more. Software QA is becoming more and more like TSA security at the airport: illogical, and obviously so. Last year, flagrantly unacceptable code was promoted without question while its replacement was held up on a meaningless detail.

We programmers are a funny lot. Make us struggle for business or technical reasons and we adapt beautifully. Make us struggle for something stupid and we just get pissed off and do something else. What a pity.

225. Can business software be life critical?

As a business programmer, I've worked on quite a few things where there is much more at stake than just money. Just a few of them:

```
- scheduling & routing of ambulances and firetrucks
- scheduling & routing of trucks carrying time-sensitive medical supplies
- clean-room quality control of medical devices
- distribution of pharmeceutical formularies
- medical claims processing & adjudication
- formulas & recipes for large batch food processing
- medical demographic databases of allergies
- distribution of mission critical airline parts with linked certifications
- certification of automotive safety devices, including airbags
- building contractor specifications, including electrical & plumbing
- clinic scheduling
```

Just because something won't hurt you immediately doesn't mean that it can't hurt you "eventually". You can see from my examples that so much we program does affect the welfare of many, even if indirectly.

We really have reached the point where software QA is just as important as engineering QA. We programmers aren't the only link in the chain, but we are an important one.

I have looked at horrendous enterprise code that supported critical health and safety issues and thought, "Do you really want to get on that plane?" or "Are you sure you want to take that pill?" (Hopefully QA catches most of the potential culprits.)

Thanks for getting us to think about it a little more. This sort of thing should always be on any good developer's mind.

226. How risky is free software?

Here's the dirty little secret about B2B software:

It's not about the money.

It's about the risk.

I tell my customers that if they can find a free generic horizontal piece of software that provides them value without risk, then by all means, use it.

Then I ask them to consider questions like these:

```
- Is the website up?
- Is the fulfillment system up?
- Is the phone system up?
- Did we get that order?
- Did that order ship?
- Did the customer pay?
- Did the bank get the money?
- Did the material get received?
- Did we make payroll?
- Did we get the best price?
- Where are our revenues below plan?
- Where are our margins below plan?
- What are our numbers for the day? Week? Month? Quarter?
```

Then the killer question: "If we don't know the answer to any of the above because of our free software then who are you going to call?"

If you have a good answer to that question, then you may want to consider a free software alternative. Otherwise,

paying for legitimate software and support is simply the price of doing business. The risk to your business and its stakeholders is simply not worth the couple of bucks saved on free software.

227. What questions would you ask a new boss?

1. Do you enforce policies uniformly? Allowing some customers to bypass channels may seem helpful, but once everyone becomes special, no one is special.

2. Do you set priorities and avoid changing them? Nothing ruins morale like the emergency du jour.

3. Do you stick up for your programmers? Blaming them for your problems is an awfully quick way to lose them.

4. Are you consistent? If you tell customers one thing and programmers another, eventually they will all realize that you're the problem.

5. Do you understand the customers' business well enough to set priorities? No one wants to work on Important Problem #42 when we're going out of business.

6. Do all policies and procedures take into account the importance of production and dev environments, security and audit controls, quality, and testing? Programmers shouldn't have to educate you why this stuff is important.

7. Are you willing to do what you ask programmers to do? Working late is fine until you do it 18 nights in a row while the boss is at the bar across the street.

8. How good is your BS filter? Decisions made based on bad data will come back to haunt us all.

9. How "professional" are you? No matter how tough things get, most programmers don't want to hear your scream, cuss, or throw things around. Better to just leave and let us fix it.

10. Do you buy us lunch once in a while? Amazing how much harder we'll work for a sandwich and a smile.

228. Why do enterprises stifle creativity?

Just a few enterprise axioms:

```
- Stuff must work consistently, regardless of who works there.
- All data and programs must satisfy corporate audit requirements.
- All data and programs must be backed up and restorable.
- All technology must be supported, no matter what.
```

I've seen all kinds of creative solutions to enterprise problems that didn't satisfy these axioms. And you know what happens? Something goes wrong and no one knows what to do.

That server under Joe's desk with those cool spreadsheets just crashed and no one ever backed it up. The nifty little Ruby app that Sue put together; she's gone and no one else knows Ruby. The open source web server just crashed and no one is to blame because there is no vendor to blame. The nine new kinds of smart phones that different sales guys like but no one knows how to support.

It goes on and on. Creative people get creative. Cool solutions help. Until they don't help anymore. Because the costs suddenly outweigh the benefits.

And one more dirty little enterprise secret than many creative people overlook: you can solve most problems with many different tools. Sure the old tools may not be as fun, but they can still get the job done at a lower cost.

In an ideal world, we'd get to work will cool stuff AND help the enterprise at the same time. But if the trade-off is our intellectual gratification vs. the enterprise's profits, the enterprise will win every time; count on it.

229. Office Pet Peeves

1. Don't sit on my desk. I'm a programmer. My desk is also my dining table.

2. If I'm eating at my desk, don't touch my food. I'm so busy that I usually bring what I think I'll need for the day. I didn't factor in your needs too. Go to the vending machine.

3. If your come to my desk and see my typing furiously and I don't look up, that means I'm busy writing code. If the building's not on fire, go away and send me an email.

4. If you get upset in a discussion with someone else, don't raise your voice, don't yell profanities, and most of all, don't slam the door or throw anything. That's when I leave for the day. Some of us have had enough of that for one lifetime.

5. If we're talking in my office, face me and take your hand off the doorknob. If it's important, I'm not going to rush through it because you're in a hurry to get somewhere else. If it's not important, then leave me alone.

6. If we're meeting, turn off your cellphone. If it vibrates, don't look at it to see who it is. If I'm not the most important person at that moment, then I don't want to meet with you.

7. I'm always happy to discuss important matters, but I don't do status meetings. If you want to know status, email me and I'll reply. Otherwise, my status report would read, Nothing accomplished. Spent all day in status meetings.

8. If there's cake in the breakroom, have a piece, but leave your Tupperware in your car. This isn't Cheesecake Factory.

9. Don't lie to me. Ever. If you tell me that Joe agreed with these mods, I can easily confirm that with Joe. If he says you never talked to him, I will never listen to anything you ever say again.

10. If you had Mexican for lunch, cut the rest of us a break and use the restroom at the Shell station.

230. User Pet Peeves

Being told "how" instead of "what". Use this API to access that database to do this process. Don't tell me how to do something. Tell me what you want. Let me figure out how. That's my job.

Being told "what" instead of "why". So you need a 4 GB csv download of the entire database every day at noon? Why? Oh, in that case, why don't you just use which has been right at your fingertips all along. If you don't understand the system, ask and I'll be glad to help. But don't make up unnecessary work for me.

Being asked for data to put into your Excel spreadsheet which you will screw up 8 minutes later. Then you want me to fix it. Forget it. If you have a business problem, tell me about it. We will find a solution together. Lone rangers don't get help. I'm not Tonto.

Calling me every 42 minutes for 3 days asking, "Is it done yet?" No, it's not. I'll let you know when it is and if there's a problem, I'll let you know about that, too.

Not calling me for 3 weeks after a release. Either it's working perfectly or no one's using it. It would be nice to know which.

Criticizing my work in a meeting in front of others without talking to me first. Big mistake. You don't want to piss off your waiter and you don't want to piss off your programmer.

Meetings when a phone call would have sufficed.

Phone calls when an email would have sufficed.

Status meetings. Pointless. I'll email status. Any questions, click "Reply".

Code walkthroughs that are more concerned with syntax than function.

SQL SELECTs inside of iterations. Please go work at McDonald's.

Sarbanes-Oxley (SOX).

231. Do bosses lie?

Just a little personal experience:

```
BOSS:    "There will NOT be a layoff."
REALITY:  There was a layoff.

BOSS:    "There will be no more layoffs."
REALITY: There were more layoffs.

BOSS:    "This will be the last layoff."
REALITY: There were more layoffs.



BOSS:    "I will be the next person laid off."
REALITY: He wasn't.  Someone else was.

BOSS:    "I am instituting 35 hrs. pay for 40 hrs. work."
REALITY: It lasted one pay period before the layoff.

BOSS:    "The corporate jet will be the next thing to go."
REALITY: The corporate jet was still in place after the layoff.

BOSS:    "Customer XYZ will pay their bill next week."
REALITY: Customer XYZ never paid their bill.  Layoff instead.

BOSS:    "Finish this project and we're in the clear."
REALITY: The project was finished.  Then came the layoff.

BOSS:    "Indispensable employees will be spared."
REALITY: No one was indispensable.

BOSS:    "We made our numbers.  We'll be OK."
REALITY: The SEC and IRS disagreed.  We went out of business.
```

```
BOSS:    "U.S. manufacturing is solid and protected."
REALITY: 400 jobs shipped to Haiti within 90 days.

BOSS:    "A layoff will be the last resort."
REALITY: Layoff + executive auto leases still in place.

BOSS:    "I will promote you next week."
REALITY: The company newsletter reported his girlfriend getting my job.

BOSS:    "Just help me get through this and I will reward you."
REALITY: I helped him get through it.  He didn't reward me.
```

Sorry to say, moral of the story:

```
Q:  How can you tell if the boss is lying?
A:  His mouth is moving.
```

Chapter 10

Selling

232. How important is networking?

Network, network, network, network, network! (Sorry, I cannot overemphasize this.) "You" are your own marketing department. All the time. No, you don't have to be one of those Amway pests, but don't be afraid to say what you do and to volunteer your opinion about something computer related. You're not doing it to get business; you're doing it because that's who you are. Even if nothing happens now, it could 6 months later. You never stop networking, no matter how busy you are now.

A few examples:

- Hung out with the same guy at Tuesday night Bible study for 3 years. One day he said, "I heard you tell someone you know something about computers. My company needs software for our factory. Do you know anything about that?" Turned into 50K over the next 6 months.

- Went to an industry dinner/speaker event. The stranger next to me asked what I did. I told him. He asked if I ever did . Before I could answer, my partner joked, "That's how we made our first million." The stranger said, "How'd you like to make your second million?" We talked all night and started work 2 days later. 20K in 2 months. All from a joke.

- A contractor friend got a great full time job. She asked me to "take over" her maintenance accounts (3 of them). Many thousands part time over the next 3 years.

- Had another friend who I met for lunch once a month for years. She always talked about her job. One day, she suddenly had to move out of state for personal reasons. I emailed her employer, telling what I did (which was exactly what they had her doing). Turned into 4 years of work.

- Met my aunt's next door neighbor while sitting on her porch. My aunt said, "Eddie's into computers." He said he had a friend who owned a pawn shop with a computer running Windows that "froze" every day at 3:00, their busiest hour. He was going nuts. (Licking my chops), I said I could look into it. A 6 month gig with all new cool software (not Windows).

- Went to a Monday Night Football party. A friend of a friend who owned a small distribution company said the bank wouldn't lend them any more money until they computerized their inventory. After 3 months of me (for $20K), they were able to borrow $300K. Pretty good deal for everyone.

- A friend was offered a 6 month gig in Detroit for $60/hour. He didn't want to move to Detroit. I took it. Got an efficiency for $400/month, drove my own car there, and dialed in to my other clients. 6 months later, moved home. Not a bad deal.

- Had another friend who owned a small software house. (Didn't know it until I knew him for over a year). He coded everything with linked lists because he didn't know anything about databases. I converted all his software to DBMS over a 6 month period. Again, everyone happy.

I could go on and on, but you kinda get the picture. And I haven't even touched on the web stuff.

The demand still far outweighs the supply for good software. If you know what you're doing (a big assumption), there's millions of people who need what you do. So get out there and talk to them!

233. How do you find customers?

How do I find customers? I just talk to people. All the time. I allow my inquisitive nature to take over. But most importantly, I really care. A mentor of mine once told me that lots of people need what we provide, so it's our responsibility to find them and see what we can do to help.

If I'm in someone's business (even as a retail customer), I often ask to watch as they enter data into their customer facing system. This invariably leads to some discussion and who knows what else. I attend events and network regularly, even if it's just staying in touch with acquaintances and asking what others are up to. Email works really well for this.

I don't have a solution in mind because I want to listen to the other's problems first. If something I have written (generally small business or e-commerce) sounds close to what can help them, I may go down that path. Anything else, I pass, but gladly provide a referral to someone who "can" help, if I know of any.

I don't cold call in the pure sense. That's just not my nature. If I ever got good at that, then I wouldn't need to be a programmer, I'd just sell someone else's product.

234. What do you talk about with prospects?

Businesses in general are not looking to buy products or services. They are looking to solve their own problems. This is NOT a discussion about your product or service. It is a discovery of the thing that bites them in the ass (that they would do just about anything to get rid of).

As you meet people (and you need to be out there in order to do this), you'll have to let them know what you do or have in order to get the discussion going. After that, the discussion is entirely about their problem.

"What do you do?"

"We have a web service that does ."

"Interesting. We've never been able to ."

"Really? Why is that?" OR

"Really? Tell me more about that?" OR

"Really? Then why don't you . We've had a lot of success helping do that."

You get the idea.

Once you help them identify and articulate their problem, one of two things will happen, either you drive the dialogue into the next step in the process or you turn and run the other way. Either way, you both win.

235. How should I handle a 1st customer meeting

First of all, remember that this is an INTRODUCTION, no more, no less. So take the following words out of your vocabulary right now: don't, but, money, donations, profit, give, bring. You are there to get to know each other, that's all. Be prepared to speak openly and honestly when asked. And be prepared to LISTEN. These are the best things you can do for him.

I would not push anything in a first meeting, but I would be prepared to respond to any question. You don't mention whether this introduction is over a meal - all the more reason to relax, take it easy, and enjoy.

I would spend some time up front preparing. Mentally have a list of any possible question and your response. Also, learn something about the customer. You already have a mutual contact (your previous CEO), so you should have something interesting to talk about besides your project. Remember, he is interested in YOU as much as your work. Give him a chance to get to know you.

Most of all, do or say something that will make him remember you, so whenever you follow up, he'll immediately know who you are. A light discussion about the local football team or an activity one of his children is involved in may work. A hand written thank you note is always nice (Why doesn't anyone do this much any more?) But beware: you MUST be sincere in whatever you say or do or you'll look like a jerk and do more harm than good.

Most of all, have a good time! And post back to let us know how it went. I, for one, will be looking for your post.

236. How do you tell your story?

"It took me five years to figure out (a) I needed a story and (b) what the story was. It's hard. But one story beats a pile of AdWords A/B tests."

I have found that there are 2 kinds of stories: classes and instances...

Class: "X can solve problem Y using our product."

Instances:

"Acme saved $30,000 per month by figuring out how to better load their trucks using our optimization software."

"The Smith family had their first ever reunion when John and Linda Smith realized how easy our family organizing software was."

"Jones Gifts doubled their sales in 3 months using our bolt-on e-commerce solution."

The class is good. The instance is better. People love stories and the instance is a real story, while the class is the framework for a potential story. The class is a commercial; the instance is a testimonial. Also, an example cuts through all the clutter right to the reader's reptilian brain. Naturally, the closer the instance is to the reader's situation, the better. OP's story was a class. I would have loved to hear a few instances of that class: some real stories about people who got real benefit from his product. People naturally want to know about other people.

237. What Makes the Top 1%?

Selling and marketing, knocking on doors, and touting your products and services "shows you as an idle developer"?

Actually, it shows you as a go-getter, exactly the type of person I'd want working for me.

The biggest difference between the top 1% and everyone else? They never stop selling.

I've done the same thing as OP and it works. Better yet, if you have office space, invite your neighbors over for wine and cheese (or beer and soda) Friday after work. Socialize and share. They may not become customers that day, but when they (or someone they know) needs what you offer, who do you think they're gonna call?

238. Buying Cycles

"Six months later, things are still sounding great and not happening. What's going on?"

It could be that nothing unusual is going on. A six+ month buying cycle for anything over 4 figures is normal. Whenever selling to an enterprise, you should ask your contact:

1. Who is the champion?
2. Who is the decision maker?
3. What is the process for each tier?
4. What should we do the best ensure our mutual success?

The only reason for surprises in the sales cycle is if you didn't bother to ask.

It someone sold a cure for cancer for $1000, everyone would buy it and the world would be healed.

If it cost $10,000, you'd probably have to await corporate paperwork and approval for 6 months and only then start implementation.

239. The Answer is Always "Yes"

Buyers of software products, like small children, hear one word more than any other: "no". "No, it can't be done." "No we don't do that." "No, if you did that it would screw up everything else." "No, that's stupid" It doesn't matter if you're right, all that matters is that you're just another person saying "no".

You differentiate yourself from others by giving the exact same answer, but with the word "yes" instead of "no".

"Yes, in order to do that, we'd also want to look at..."

"Yes, let's make it 'pop' using some of the things we bring to the table..."

"Yes, no one even thought about that, and we should now before we get any further into this thing..."

or even the extreme:

"Yes, there's a way to do that. No one has ever done that before, so now is the time for someone to be first..."

As I've told my customers many times, "The answer is always 'Yes'. You may not want to do it once you understand what it will take, but the answer is still 'yes'."

No other word has helped me more to find myself and do my best work for others.

240. How do you crack the enterprise world?

I have been on both sides of the enterprise software sale many times and have concluded that a) it always sucks and b) it's rarely in "anyone's" best interest.

So instead of examining the current model and making suggestions for accomodating or improving it, I prefer to suggest an alternative.

I believe the best way to crack the enterprise software market is the same way to eat an elephant: one bite at a time through the soft underbelly...

Find a critical business function being done in Excel and provide an alternative web app.

Find a "business within a business" and automate it with modern technology. (Examples are small independent business units, warehouses, job shops, sample shops, "anything" a user has set up that "can" be autonomous.)

Provide a modern satelite system to augment and integrate with an existing enterprise monster. (A separate module for one function like payroll or fixed assets, special processes for marketing, engineering, manufacturing, etc.) The possibilities are endless. "Somebody" is not getting what they need out of SAP, Oracle, or whatever.

Provide a separate business unit with everything they need. This may be cheaper than the customer adding more licenses to their ERP system.

The key to this approach is staying under corporate IT's radar. The way to do that is by keeping your prices below your customer's boss's threshold.

How do I know this can work? Because it has, many times. I have implemented dozens of apps in enterprises that they thought they could never have because of the existing software and sales model.

And I remember history. At one time, IT departments were very threatened by PC's. They challenged their ivory tower with a mainframe and dumb terminals. So users just bought their own PCs from their expense budgets and forced IT's hand.

Lightning can strike twice. Users are once again tired of waiting 18 months for a fix and are ripe for a custom 37signals type of solution. Let the app rush begin.

241. What are the Spending Authority Cut-Offs?

The #1 question to ask when selling to the enterprise is, "What are the spending authority cut-offs?" Nothing means more.

Funny, in my experience, the 5K and 100K numbers are pretty accurate.

I recently assisted in the purchase of business intelligence package. The CIO (my contact) had authority to spend up to $100K. Anything more had to go the board, and that "just wasn't going to happen". One vendor knew what to ask and bid $93K. The two others were much higher. Guess who (automatically) got the sale. The other 2 may have been better, but we'll never know. They were effectively eliminated by rules they never asked about.

At the low end, almost everyone has authority to spend up to $5K, even users. They bring in desktop software or SAAS under IT's radar. There's "huge" demand for solutions to their problems that fit under their spending limits.

242. How do you close the deal?

You have to get your prospects to think that what you're offering was their idea all along.

How do you do this?

Get to know them. Spend time with them. Find out what their lives are like, what they have to go through to compete, and what makes them suffer. Jump into their pool at the deep end and learn how to swim. Walk through their warehouses, customer service departments, and general offices. Sit down at their computers and try to do their jobs. Get them talking.

Once they see that you are sincere and have something to offer, they will not be bashful. They will tell you everything you need to know to help them. This will do 2 critical things:

1. It will provide specific feedback about what you're building or have built, whether or not it makes sense for them, and what to change/fine tune/refocus. It they need it like that, chances are that many others do too. Your first prospects have unwittingly been the best focus group you could have assembled.

2. You will be offering exactly what they asked for so they will have few excuses not to buy. Do not underestimate the solid gold of this approach; it works incredibly well.

Call this good sales and marketing if you want but I never have. I just call it doing whatever it takes to help your customers. Becoming successful is a byproduct.

243. How do I close a sale?

"How to close a sale?"

Ask your customer, not us.

I'm not trying to be abrupt, but it sounds like you've already done all the right things and your relationship with your customer should have reached the point where you can ask them exactly this question.

Dealing with institutions can be it's own animal. The best way to learn how their buying process works is to ask them! In a perfect world, you may still be 6 months away from a sale. You wouldn't agonize over it if you knew, and you'd know if you asked.

In dealing with institutional customers, I even take it a step further. Before I invest any time in the sales cycle, I have them teach me what it takes to get a sale, exactly what I have to do, and how long it will take. Real buyers will be happy to tell you all of this; in fact, they may think you're sales amateurs if you don't bother to ask. Lots of times the buyer may be frustrated by their own organization and will coach you to be more successful so that they can get what they want.

There are millions of potential tips: "You have to talk to Joe Smith first." "Never call Fred on Monday." If you filled out Form XG7-B first, you'll save 6 weeks." "Mary only buys from people she meets through Bill or her Business Group." "You have to be a preferred vendor of XYZ.."

I hope you get the picture. Like I said, it sounds like you've done all the right things so far. No one here at hn knows what else you need to do. Your customer does. Ask them. Today.

244. The One Excuse Not to Network

I have found most networking (of any kind) to be an inefficient use of my time. At most events, I always had a little voice in my head saying things like, "Instead of being here, I could be building ," or "What could possibly come out of this discussion?" I'm also frustrated because so many events don't have my prospects, but "people who know people who know people who may know a potential prospect of mine".

I have taken a totally different approach. It's really simple and maybe even counter-intuitive. Hear me out:

Be excellent. Better yet, be "very" excellent. In everything you do.

If my customer doesn't think I'm their best vendor, then I have failed.

This applies to "everything". In the work that I do. In the products I supply. In the fun their people have with me. In the "outside their box" thinking about every project. In the communication. In the failsafe processes of doing business (Yes, I double check that some has double checked.) In thinking 2 steps ahead of them. In being a trusted partner in that part of their business. In pristine ethics (Don't underestimate this one; one slip neutralizes "everything else you've ever done".)

When I conduct business this way, I become a magnet to those who need my services. I call this "passive networking". I spend no time networking, no time marketing, pay no referrals, and focus completely on my customers. They know and appreciate this. When one of their colleagues mentions a concern at "their" networking meeting, their Tech Club, their restaurant, or in a discussion with their vendors and customers, they think of me. When they care about the people they know, they want the best for them. I always want to be thought of in this way. IMO, "this" is the definition of totally efficient marketing.

I know it sounds awfully old school and like a cop-out, but doing everything I can to make myself a magnet is the best thing I ever did for my business. So instead of wasting 99% of my time with strangers, I spend it directly investing 100% of it in people that already matter.

Chapter 11

Just for Fun

245. What was your most creative resume?

If it worked for Leonardo da Vinci, maybe it could work for me. The next time I'm looking for a job, I'll try this:

"Most Illustrious Proprietor, Having now sufficiently considered the specimens of all those who proclaim themselves skilled developers of applications of business, and that the invention and operation of the said programs are nothing different from those in common use: I shall endeavor, without prejudice to any one else, to explain myself to your Company, showing your Management my secret, and then offering them to your best pleasure and approbation to work with effect at opportune moments on all those things which, in part, shall be briefly noted below.

1. I have a sort of extremely light and strong functions and modules, adapted to be most easily ftp'd, and with them you may pursue, and at any time combine them with others, secure and indestructible by standard mean time to failure of hardware and denial of service, easy and convenient to compile and catalog. Also methods of unzipping and storing the data of the customers.

2. I know how, when a website is besieged, to shard data onto the cloud, and make endless variety of mirrors, and fault tolerant disks and RAIDs, and other machines pertaining to such concerns.

3. If, by reason of the volume of the data, or the structure of the btrees and its indexes, it is impossible, when conducting a search, to avail oneself of sub-second response time, I have methods for benchmarking every process or other function, even if it were interpreted, etc.

4. Again, I have kinds of functions; most convenient and easy to ftp; and with these I can spawn lots of data almost resembling a torrent; and with the download of these cause great terror to the competitor, to his great detriment and confusion.

5. And if the processing should be on the desktop I have apps of many machines most efficient for data entry and reporting; and utilities which will satisfy the needs of the most demanding customers and users and consumers.

6. I have means by secret and tortuous scripts and modules, made without leaving tracks, to generate source code, even if it were needed to run on a client or a server.

7. I will make secure firewalls, safe and unattackable, which, entering among the hackers with their utilities, there is no body of crackers so great but they would break them. And behind these, software could run quite unhurt and without any hindrance.

8. In case of need I will make big properties, methods, and collections and useful forms, out of the common type.

9. Where the operation of compiling might fail, I would contrive scripts, functions, routines, and other parameter driven processes of marvellous efficacy and not in common use. And in short, according to the variety of cases, I can contrive various and endless means of data entry, reporting, and storage.

10. In times of low revenue I believe I can give perfect satisfaction and to the equal of any other in maintenance and the refactoring of code public and private; and in guiding data from one warehouse to another.

11. I can carry out code in Javascript, PHP, or C, and also I can do in network administration whatever may be done, as well as any other, be he who he may.

Again, the intranet app may be taken in hand, which is to be to the immortal glory and eternal honor of all your customers of happy memory, and of the illustrious house of Google.

And if any of the above-named things seem to anyone to be impossible or not feasible, I am most ready to make the experiment in your data center, or in whatever place may please your Businessperson - to whom I comment myself with the utmost humility, etc."

246. Hacker News Front Page 12/31/2019

Hacker News 12/31/2019 new | comments | leaders | jobs | submit login

1. Tell HN: Congratulations Patio11 - first to reach 1,000,000 karma
   4 points by iamelgringo 1 hour ago | discuss

2. Ask HN: Any Predictions for the Year 2029?
   11 points by DanielBMarkham 37 minutes ago | 8 comments

3. The Apple Tablet to Launch 1st Quarter 2020 (cnet.com)
   210 points by vaksel 20 hours ago | 122 comments

4. President-Elect Graham to Appoint Sam Altman to Cabinet (msnbc.com)
   14 points by muriithi 4 hours ago | 2 comments

5. Trevor Blackwell's Robot Collects Rocks on Mars (science.com)
   143 points by ojbyrne 18 hours ago | 81 comments

6. Tell HN: Hacker News is getting too much like reddit
   17 points by jamesjones 6 hours ago | 3 comments

7. Last Land Line Disconnected at Midnight (cnn.com)

6 points by chickamade 3 hours ago | discuss

8. Mark Zuckerman buys Portugal (worldnews.com)
   51 points by larryz 14 hours ago | 16 comments

9. How Half Our Staff Telecommutes from Space (joelonsoftware.com)
   45 points by jspolsky 13 hours ago | 2 comments

10. No Deadlines Needed After Singularity is Reached (wired.com)
    44 points by bxgame 14 hours ago | 28 comments

11. Ask pg: Why do YC teams only get $1,000,000?
    19 points by abcklm 9 hours ago | 5 comments

12. KidneyExchange.com has 10,000th successful transplant (yahoo.com)
    23 points by phsr 10 hours ago | 7 comments

13. Walmart Acquires Microsoft (wallstreetjournal.com)
    76 points by francis24 20 hours ago | 17 comments

14. Baby Communicates from Womb via usb23.7 (scientificamerican.com)
    13 points by johnson 8 hours ago | 7 comments

15. Mark Bao Starts 1,000th Start-Up (startupnews.com)

4 points by MarySmith 3 hours ago | discuss

16. unalone accepts Pulitzer for blog (cnn.com)
    20 points by bootload 10 hours ago | 11 comments

17. Ask HN: Review my app: NoMoreAds.com (nomoreads.com)
    17 points by fred 10 hours ago | discuss

18. Poll: Favorite Language, Ruby 92.7 or C++++++++
    37 points by uafes 17 hours ago | 5 comments

19. Feds Force Google to Divest its Apps Business (news.com)
    38 points by pete 17 hours ago | 5 comments

20. Burrito Tunnel Between Calif & NYC Finally Completed (onion.com)
    50 points by jose 20 hours ago | 20 comments

21. In 2020 Belize will become the world's second-largest economy (economist.com)
    30 points by pg 16 hours ago | 23 comments

22. Ask HN: What was Microsoft Office?
    63 points by yahfsh 23 hours ago | 6 comments

23. Wikipedia Available on Gumwrapper (abc.com)
    3 points by lapenne 3 hours ago | discuss

24. Boeing Dreamliner Delayed Until 2022 (airlinenews.com)
    4 points by mitchel 5 hours ago | discuss

25. Ted Williams becomes 1st to win MVP with 2 different bodies (mlb.com)
    5 points by johnson 6 hours ago | 2 comments

26. Ask HN: Review my app (virtualsex.com)
    125 points by ghpoa 1 day ago | 13 comments

27. Science: Cigarettes Were Healthy After All (science.com)
    43 points by woodyallen 20 hours ago | 14 comments

28. Broadband Finally Reaches Flint, Michigan (cbs.com)
    133 points by johnguest 1 day ago | 20 comments

29. GO TO Added to Python, 27 Programmers Jump Out Windows (python.org)
    149 points by swert 1 day ago | 20 comments

30. Wipe The Slate Clean For 2020, Commit Web 9.0 Suicide (techcrunch.com)
    2 points by nreece 2 hours ago | discuss

More

Lists | RSS | Bookmarklet | Guidelines | FAQ | News News
Feature Requests | Y Combinator | Apply | Library

247. The Programmer's Aptitude Test

(Don't scroll down until you're done.)

1. You push your cart through the supermarket
   a. In a pre-defined manner
   b. Randomly

2. When watching football on TV, you focus on
   a. the quarterback
   b. the defensive linemen

3. You drive to work
   a. the same route every day
   b. with a different route every once in a while

4. Which card game do you prefer?
   a. bridge
   b. poker

5. To plan for tomorrow's weather
   a. You check the TV or internet.
   b. You go outside, looking for signals.

6. Who do you prefer?
   a. Andrew Carnegie
   b. Marie Curie

7. You prefer
   a. your keyboard
   b. your mouse

8. Which subject do you prefer?
   a. history
   b. literature

9. Which would you rather do?
   a. take a walk in the woods
   b. a crossword puzzle

10. Which is more important to you?
    a. time
    b. space

Answer: If you tried to figure out (game) the test as you took it, you have a programmer's aptitude. If not, you don't, and probably don't even understand this answer.

248. If Samuel L. Jackson were a Programmer

"The path of the righteous programmer is beset on all sides by the inequities of the clueless and the tyranny of evil project managers. Blessed is he, who in the name of achievement and solid technology, shepherds the users through the valley of ineptitude, for he is truly his customer's keeper and the finder of lost solutions. And I will strike down upon thee with great vengeance and furious anger those who would attempt to deploy without testing. And you will know my name is zedshaw when I lay my software upon thee."

249. The Thread Not Traversed

Kinda reminds me of the time Robert Frost was a technical writer for a day :-)

```
The Thread Not Traversed

TWO threads diverged in a yellow stack,
And sorry I could not traverse so
And be one parser, long I stood
And looped through one as long as I could
To where it spawned in the overflow;

Then invoked the other, as just as fair,
And having perhaps the faster stats,
Because it was hashed and wanted wear;
Though as for that the data there
Had run them into third normal form,

And both that session equally lay
In objects no instance had trodden class.
Oh, I kept the version for another day!
Yet knowing how code leads on to code,
I doubted if I should ever unit test.

I shall be documenting this with a sigh
Somewhere many transactions hence:
Two threads diverged in a stack, and I—
I took the one less traversed by,
And that has caused the site to fly.
```

250. Programmers are Practical

A salesman, a project manager, and a programmer are kidnapped by terrorists on the way to a customer demo. The company refuses to pay ransom so they are to be executed. The kidnappers grant each a last request.

The salesman said, "I have been working very hard on a Power Point presentation of our new release and I haven't had a chance to present it yet. It's only 143 slides and 2 hours long, and I'd like to present it before I'm killed."

The project manager said, "I have developed a new methodology for implementing our new release. I'd like to present 25 flip chart pages to describe it. I will only take one hour."

The programmer said, "Kill me first."

251. What if I miss a day of Hacker News?

I got invited to an interview, what'll I do?

I got rejected, what'll I do?

PHP sucks.

C# sucks.

Microsoft sucks.

Ruby rules!

Best. Interface. Ever.

Scientists in Sweden produce Britney Spears in a petri dish.

pg is great.

Where can I get the best corporate lawyer in the world for free?

Anyone want to buy some legal drugs?

Ask HN: When I sit too long in this chair, my back hurts. What should I do? HN: Switch chairs.

YC should start a franchise in Europe.

YC should start a franchise in Latin America.

YC should start a franchise in Asia.

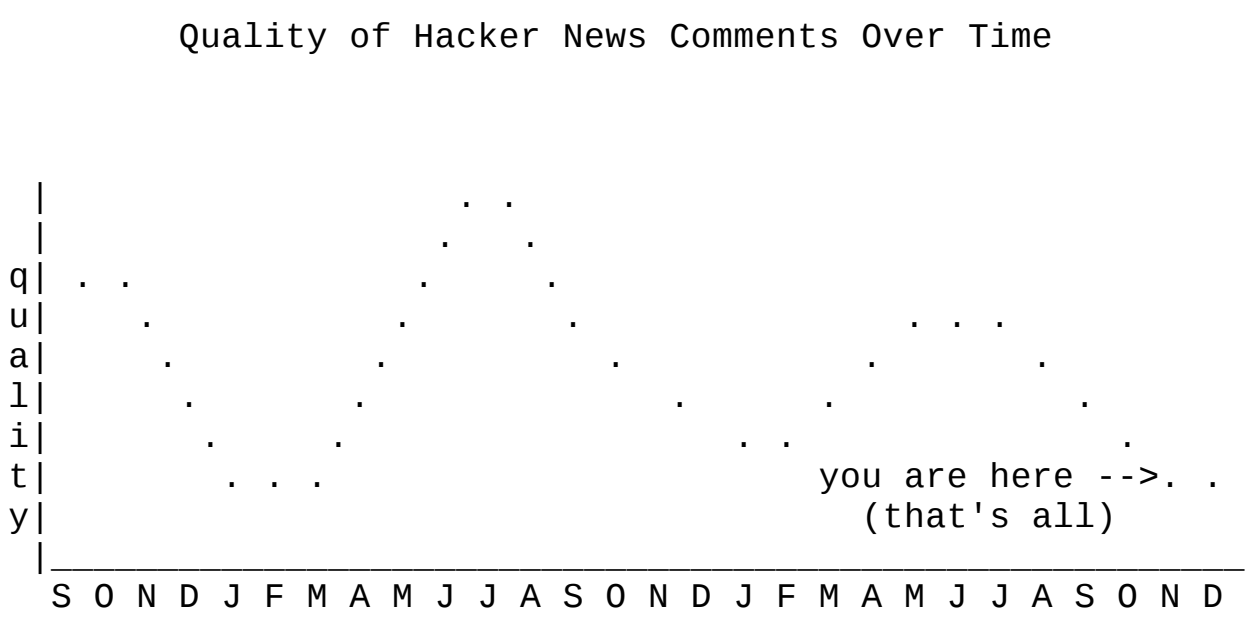
YC should start a franchise in Brooklyn.

Ask HN: I don't know how to code? How can I build a web app that makes a million dollars by Tuesday?

Ask HN: I need motivation. Any hints?

There. You just read all of today's HN news. Now close your browser and get to work.

252. Quality of HN Comments Over Time



253. Some Things Can't be Duplicated On-Line

Girlfriend: Your wardrobe needs updating. (I love you but you look like shit.)

Me: No it doesn't. (Yes it does, but I have no idea what to do.)

Girlfriend: There are some great sales at the mall. (I'll appeal to his sense of financial responsibility.)

Me: I'm too busy right now. (I really am busy and I hate going to the mall.)

Girlfriend: You would look so much better in a palette more suited to you. It seems such a waste to not take advantage of your dark eyes and olive complexion. (I'll appeal to his vanity.)

Me: Wait, what? Dark eyes? Olive complexion? (She may be on to something.)

Girlfriend: Everyone thought you looked so cute in that shirt I got you last month. And the haircut, too. All my girlfriends thought you were so hot. (This works on ALL men...)

Me: No they didn't. (Hmmm...her girlfriends think I'm hot? Whoa.)

Girlfriend: And you can't go to that party in the same jeans and t-shirt again. Every picture of you on Facebook is in the same outfit. Potential investors might think you're a flake. (Threaten his start-up; threaten his baby.)

Me: OK, OK. (Good point, I keep hashing to the same frame.)

Girlfriend: I'll get dressed. You get the car. We can stop for a couple of drinks to celebrate on the way home. I'll probably love how you look. (Moving in for the kill...)

Me: This is so superficial, but you win. (Thank you, thank you, thank you.)

There are still some experiences that just can't be duplicated on-line.

254. A Time to Work and a Time to Play

edw519's thoughts about workaholism (with apologies to the original author(s)):

To every thing there is a reason, and a time to every purpose under your project

A time to study and a time to write

A time to code and a time to pluck up that which is coded

A time to kill ideas and a time to heal that patch

A time to break down algorithms and a time to build up frameworks

A time to weep about bugs and a time to laugh about clean compiles

A time to mourn that dead end and a time to dance when it works

A time to cast away duplicates and a time to gather common functions together

A time to embrace someone else's code and a time to refrain from embracing it

A time to seek advice and a time to lose illogical prejudices

A time to keep and a time to refactor

A time to clean up variable names and a time to rewrite

A time to accept and a time to keep testing

A time to love your idea and a time to give it up

A time for plowing onward and a time to rest.

255. A Programmer's Poem

My friends all think
  that I'm a neb
Cause I spend much time
  here on the web

The good things here
  I do not abuse
Except lots of time
  on hacker news

I don't read reddit
  I will not digg
I'm not on facebook
  My work's too big

I do not text
  I do not tweet
I just work on
  Things that are neat

I check email
  throughout the day
But there are no games
  that I will play

My phone's on vibrate
  I do not chat
My work is really
  Where it's at

Knuth and Turing
  are my big heroes
I love to move
  Ones and zeros

My head is down
  I'm in the mode
Don't bother me
  I have to code

Those who need me
  leave voicemail
I'm much too busy
  trying not to fail

I learn on-line
  and from my schools
But I must avoid
  all sorts of trolls

I can't believe
  I wrote this ode
When I have so much
  I have to code

I'm not an addict
  I have no drug
I've got to go
  To fix a bug

256. Little Known Development Methods

Garbage Perpetuation Development (GPD) - You can't believe how bad the existing code base is, but you're afraid to open a can of worms, so everything you add to it is written in the same style. For the rest of your life, you can say, "It was like that when I got here."

Mansion in the Quicksand Development (MQD) - The opposite of Garbage Perpetuation Development, you are so shocked by the poor quality of the existing code that you vow that you'd rather swallow razor blades that code the same way. So you write a tight beautiful refactored masterpiece that will crash as soon as the underlying database loses its integrity (later tonight).

Defer to the Framework Development (DFD) - You're not sure how to tackle quite a few critical design/architecture issues, so you convince your boss to adopt the framework du jour and decide to "let it handle it". As soon as someone needs something that the framework doesn't handle, you blame management for making such a myoptic technology decision and say that it can't be done. You keep your job and get a new boss every two years.

Not Invented Here Development (NIHD) - The opposite of Defer to the Framework Development, as soon as you discover something the the current framework can't handle, you abandon it and write all you own routines. Everything now works exactly as you want it, but with all the additional code to maintain, your backlog has just grown from 6 months to 2 years.

Whoever Screams Loudest Development (WSLD) - Just as the name implies, you work for the customer who screams the loudest. If anyone screams louder, you drop everything and work on their project.

F-Bomb Development (FBD) - Whenever everyone is screaming so loud you can't hear anything, you work on the project of the customer who drops the most F Bombs.

Start Over Development (SOD) - A critical requirement cannot be supported by the current architecture, so you decide to rewrite it. You spend 3 months designing the new architecture and then 6 months writing the new code.

You never finish because you're out of business. Now you know what "critical" means.

Workaround Development (WAD) - The opposite of Start Over Development, you can make the current system do anything. You are so clever with your extra algorithms, functions, and data bases. Even with all your great variable naming and comments, six months later, you have no idea how anything works.

Code Generation Development (CGD) - You're so tired of writing the same code over and over, that you write a code generator to do it for you. What used to take a week only takes a few hours with the new tool. But you're no further ahead because 80% of your time is needed to enhance and maintain the code generator.

Infinite Prototyping Development (IPD) - Your customers and users are unable to describe or document their requirements. So you spend lots of time with them understanding their business and when you're ready, you throw together a prototype. They love it, but it needs just a few changes. You keep making changes, but it always needs more. It stays a prototype forever. When the app crashes because of security or scaling issues, you're off the hook because, "It's only a prototype."

Infinite Analysis Development (IAD) - You never have to do anything because you never have specs. Woo hoo!