The End of IT Slavery

Shlomi Fish <shlomif@shlomifish.org>

Revision History

Revision 4872	2011-06-05	shlomif
Convert single and double quotes from ASCII to Unicode.		
Revision 1746	2007-05-01	shlomif
Corrected many errors courtesy of a reviewer from the IRC.		
Revision 1742	2007-05-01	shlomif
Corrected many things, added more bolds, and a few extra text for completeness.		
Revision 1740	2007-05-01	shlomif
Finished the article.		
Revision 1705	2007-04-17	shlomif
Forked the template from a previous work and working on it.		

Table of Contents

Introduction

This is the year 2007. This is Shlomi Fish, **a good hacker**, where hacker is a good enthusiastic programmer, not necessarily a computer intruder. And I have an announcement to make: **I refuse to be an IT slave**. Moreover: if you want to employ people like me (and you do), you should not give us only good conditions - **you should give us exceptional ones**. Otherwise, we’ll probably leave, or be fired, much to your misfortune.

Some Facts about Working

Do you honestly believe that if you employ someone for 9 hours, you'll get an average of close to 9 hours worth of code writing a day? Probably not. Programmers tend to get distracted. They find it hard to start coding. They need to think about what they do. They need to take breaks for various things (like drinking, eating, talking with their co-workers, etc.).

Furthermore, actual **code writing is not the most productive activity**, as surprising as it sounds. That's because if one writes code exclusively for too long, his mind will run in circles and he'll lose his edge. And there's something that is even more productive than actually producing output.

That thing is **revolutionary decisions**. Decisions that make one more productive in the future, restructure one's code (or one's text or whatever) in completely better ways, or allow the worker to do something in a better way. The more such decisions he reaches, the more productive he is, and the more value he is to his company. He's no longer just a code monkey - he's a "Rosh Gadol" and innovative programmer [http://www.joelonsoftware.com/items/2004/12/06.html], who is of much more value to the company than just someone who writes the code that other people told him how it should behave.

Achieving Productivity

If you want your programmers to be as productive as possible, make sure you hire good programmers, who can reflect on what they do, learn more, and who are enthusiastic about writing code and like it. Only they can be of value to you.

Paul Graham called these developers "Great Hackers" in an essay he wrote [http://www.paulgraham.com/gh.html]. As I said before, I can testify I'm a good (or "above-average") programmer (and other things [http://www.shlomifish.org/]), but not that I'm a great one. That's because the baker cannot testify for the quality of his dough. However, I know some people whom I can tell are great hackers.

Who are they? They are people who come up with wonderful ideas. Who are interested in many things inside or outside computers. They often can write a lot of good code, but often their code is wonderfully ugly, but still very functional, mostly bug-free, feature-rich and efficient.

Re-use instead of Start-over

Another important factor, that is not always present in nascent programmers, is their desire or even necessity to re-use code, however ugly it may seem. Eric Raymond wrote about it in "the Cathedral and the Bazaar" [http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/ar01s02.html]:

> So, did I immediately launch into a furious whirl of coding up a brand-new POP3 client to compete with the existing ones? Not on your life! I looked carefully at the POP utilities I had in hand, asking myself "Which one is closest to what I want?" Because:
>
> 2. Good programmers know what to write. Great ones know what to rewrite (and reuse).
>
> While I don't claim to be a great programmer, I try to imitate one. An important trait of the great ones is constructive laziness. They know that you get an A not for effort but for results, and that it's almost always easier to start from a good partial solution than from nothing at all.

Later on Joel Spolsky expanded on it in "Things You Should Never Do, Part I" [http://www.joelonsoftware.com/articles/fog0000000069.html], and "Rub a dub dub" [http://www.joelonsoftware.com/articles/fog0000000348.html], and it's also been extensively documented elsewhere.

If your programmer is keen on labelling a code as “ugly”, “unusable” or “I’d like completely rewrite it from scratch”, then he’ll waste your time and be unhappy and under-productive.

Instead he should say: “give me some time to whip this code back into shape, by refactoring it [= cleaning it up]”. Believe it or not, but refactoring actually saves time, as Joel demonstrates in “Rub-a-dub-dub” and Martin Fowler in his “Refactoring” book.

So if you hire a beginning programmer with a good potential who’s full of this (bad) attitude, give him or her this reading list. It will be faster and more fun to improve the quality of the code, rather than to rewrite it from scratch.

“We Can’t Find Good Programmers”

How many times have you heard this phrase? I’ll tell you why you can’t find them. It’s because you treat them like dirt. Great Hackers [http://www.paulgraham.com/gh.html] are exceptional people: they won’t work for you, unless they’re completely happy. Some people who read the original article by Paul Graham, thought that they won’t fix bugs or do routine tasks. That’s not true - they will and often stay up late to do that. But they refuse to take abuse.

So if you want to hire great developers, you’d better make sure you offer them the best. Not only that, but you’d better make sure you recognise a great developer when you see one.

How to find Great Developers?

Open Source

Great Developers experiment on their own - they love to “hack”: create new things, enhance existing things, and make the elements of nature do things God did not intend them to do. No, most great developers don’t intrude into other people’s computers. And most of the so-called “script kiddies” (at least those who are computer intruders) while usually being perfectly nice people, are not great developers - at least not yet. Great developers instead spend hours on end improving the quality of existing code by simple, mechanical actions. They love learning new languages. They chat endlessly with people from all over the world. They read a lot of things online and offline.

Most importantly, great developers in this day and age work on open-source projects [http://en.wikipedia.org/wiki/Open_source]. While open-source became popular with software, it later expanded to many other fields: there’s now open-content/free-content texts, pictures and photographs, music, videos, and anything else. There are open patents, open science, open access, etc. But let’s focus on open-source code.

If you want to hire a great developer you can bet your life, that he’ll contribute for an open-source project. Why? Can you imagine him starting his own shareware project? Shareware is practically dead on Linux and other UNIXes, which are the development environment of choice for most people. And shareware doesn’t pay: you work on a shareware program a lot, then you release it for a pay, receive pay from at most 10 people (if you’re lucky), and no-one can or is willing to contribute back. On the other hand, if your program is open-source, and you publicise it on freshmeat.net [http://freshmeat.net/] or a similar site, some people will gladly take a look, some of them will try it (instead of immediately ignoring it for being non-open-source), some of them will send you a good input, and some will even become contributors.

Naturally, most starting open-source projects amount to nothing. But most shareware programs have died along the way too. If you have a good discipline and want your program to succeed, nowadays open-source is what all the cool kids are doing, and practically no one will respect you if you’re just a non-open-source shareware (or even just non-open-source “freeware”) developer.

So if you want to hire good developers, your best bet is to find people who are free software enthusiastic. Other people are just either not good developers, or have the wrong character, which means they will not improve, but become worse in time. Technology is progressing and workers who are not open (literally), cannot keep up with it.

Language

It is known that great programmers almost consistently have very good lingual skills. All of the great programmers I know have a very good level of English. The famous computer scientist Edsger Dijkstra [http://en.wikipedia.org/wiki/Edsger_Dijkstra] once commented that he preferred to hire English majors over Computer Science majors because the former made better programmers. If you see someone with an obviously broken English - don't hire him! (I'm not talking about a few problems - these are common and people with an audible cognition are more susceptible to them.)

Philosophy

So language is one element. Another element is philosophy. I'm not talking about the standard ivory-tower philosophy, which is mostly either useless or completely harmful, but rather of the process of "loving-the-wisdom" and seeking wisdom. Nowadays, most of the greatest philosophers don't write books. Instead they write essays and articles online, emails, weblog entries or comments, or even cartoon strips or other art forms. The great philosophers of today are bloggers.

Some great programmers I know purposely maintain a low profile and don't comment on everything. I can respect that. Others (including the writer of these lines) are completely sincere and always express their mind.

If you want to tell if someone is a great developer - ask him about his opinion on a few things. Possibly political. Possibly related to software management. Possibly something less serious, like popular culture. If they're a great developer, they'll eventually have something innovative and opinionated to say about something. Not necessarily an opinion you'll agree with, but a good, well-thought opinion.

When asked why he changed his opinion from the day before, Mahatma Gandhi replied: "Yesterday I was more stupid.". And indeed, you'll see that the really good programmers may eventually change their opinions.

How to treat Great Developers

So you want to hire a great hacker. That's great! What should you do? Great hackers, are not too concerned with getting an exceptional salary [http://www.catb.org/~esr/writings/cathedral-bazaar/homesteading/ar01s19.html]. They probably won't work for free, but they will often prefer a better job with a smaller pay than an abusive job with a higher pay. But what do they like?

Great Developers like to **work on stable, well-proven platforms** that are preferably open-source. In today's world it means either C or C++ using gcc and g++; Perl, Python, Ruby and to a lesser extent PHP; Java and possibly also .NET on Windows 2003 or the open-source and cross-platform Mono [http://www.mono-project.com/]. If your platform doesn't work, has bugs that delving into the source won't reveal (if the source can be read at all) - they're not going to be happy.

At a workplace I worked for a few days, I was given a task to automate a buggy Hebrew windows application written using PowerBuilder, which was incredibly quirky. It took me three or four days to write less than a hundred lines of Perl code. I was never so unproductive. After answering the wrong question in an interrogation, I was fired, and was heavily relieved. Such applications should not be tested - they should be rewritten using a more modern and less quirky technology.

Another thing great developers hate is to **forced to be consumed by their work**. In Israel, some workplaces require at least 9 hours of daily work. How can you work like that, and still work on open-source projects, have a girlfriend or a boyfriend, go to the meetings of social or technical clubs, hang with your friends in coffee shops, restaurants, or pubs, and other activities like that. All of these things are not **strictly part of work**, but they are what **make great developers productive**. That's because coding happens in your mind, not in your editor or IDE, and because the more inspired a developer is, the more happier he is, the better code he writes, and the better ideas he has. A programmer who just works all week, without leisure time is heavily underproductive.

Here's another thing: in Israel the law mandates at least 12 paid vacation days per year. At my previous workplace, I received 14 paid vacation days? Thank you, Workplace!!

But guess what? Fog Creek software [http://www.fogcreek.com/], best known as the company co-owned by Joel Spolsky from the "Joel on Software" site [http://www.joelonsoftware.com/] gives people 6 weeks of paid vacation annually. And they're doing very fine, and people there are super-happy and productive. So, why should I work for you if all I get are 14 stupid days? Get real!

The Best Equipment Money can Buy

The Joel Test [http://www.joelonsoftware.com/articles/fog0000000043.html] specifically says that you should give your developers the best tools and equipment money can buy - from hardware to software, to office conditions, to food, to location, to everything else. Many workplaces don't understand that.

In a previous workplace of mine, I was given a recycled computer - it was still fast but its hard-disk was only 40 GB. It was a pain to get some Windows XP virtual machines running there. Another computer we developed on still had a 40 GB hard disk too. We ran out of space there because we had a huge checkout of the Subversion version control system [http://subversion.tigris.org/] there. And the only hard-disks that our lab had were whopping 80 GB ones, bought because they were the cheapest ones, which probably wouldn't have been sufficient for too long, either.

The most amazing thing was that the computer's sole CD drive, was just a CD drive - no CD writing, no DVD reading or writing - a CD drive. It gave me a lot of grief. I had to copy a few DVDs I burned at home on my supervisor's computer, and copy them over the network, and could not install a Linux distribution from its installation DVD even if I wanted to.

Some people can still happily use old hardware or play with relatively buggy software. But good developers should not be concerned with this. They want a hardware with enough space, RAM, and processing power. They want a screen that they can easily move around (a flat-panel LCD screen). They want good food in the kitchen. They want talkative, interesting and benevolent people to talk with. They want a spacious office. They want Internet connectivity [http://www.fysh.org/~katie/computing/no-net-access.txt] so they can surf the web and search for answers, chat with their friends, ask people on the IRC for help, and contact some people who know more about the software they are using than they do.

Leave Your Developers Alone

At a previous workplace of mine, my bosses **did not like the fact that I sometimes played** card solitaire games or Sokoban. I'm not much of a gamer and don't spend hours on end playing high-end games. But they didn't like that people who passed by **saw me not working at that very moment**.

I needed these games to bring my mind back into cycle. I was also labelled as a "strange bird", because I often stood and looked at our building's garden and lawn through the window. As Paul Graham notes [http://www.paulgraham.com/opensource.html] there's a large difference between **"pretend work"** and **"real work"**. Playing games and looking through the window do not make you less productive. If you're just writing code for the same codebase all the time, your mind will soon run in circles, and you won't be productive.

In a mission statement to an innovative software company [http://tech.groups.yahoo.com/group/hackers-il/message/4784], I said that I expected developers to work for only 20% of the time? Why? They:

1. Usually won't work more anyhow.

2. Since so little is expected of them, they will feel willing to work more than that. (Assuming they are indeed great developers with a wonderful character).

3. The things they'll do in the rest of the time will inspire them and allow them to be more productive.

Another issue are **the working hours**. Make sure your developers can normally come to the office when they want and go when they want. If you sometimes need more time or better attendance, then great developers will be happy to comply - temporarily. But crunch mode is a recipe for disaster [http://www.igda.org/articles/erobinsonncrunch.php] - your developers will be over-worked, under-productive, and unhappy. And if they're smart, they will soon quit.

And here's another anecdote: at a previous workplace, I was instructed to fulfil my hours quota at least precisely, so they can get enough benefits from the Israeli Chief Scientist. But what is more beneficial for them: a happy, productive developer who does very good work, or a few extra shekels? I can never understand this skewed logic.

Be Honest with your Developers

The first thing about being honest with your developers is **telling them exactly why they weren't hired** if this was indeed the case. Most companies I've been to, either rejected me after an otherwise seemingly successful interview, or even sent me an uninformative rejection letter before any interview. This will cause the exceptional developer to wonder what has he done wrong, or what's wrong with him.

As an employer, you should have the **minimal decency to inform the star developers what they did wrong** and how to further improve. Otherwise, you're not being fair with them, and they also may tell all their friends how pointless it was to try to apply for you. (Or tell their friends how good a different company that's also looking for great hackers treated them.)

Honesty is also **telling your developers when they did something right**, like finding bugs, fixing bugs, having good ideas, implementing something on schedule, handling a situation properly, etc. and when they did not do something properly. Even the greatest developers make mistakes, but you should be honest enough to evaluate their total performance so far and not just their isolated mistake. A great developer who's worked for you for a few months, made a mistake recently learned from it, and is determined not to repeat it, **is a better asset than a new developer** who still has a lot to learn at the company.

Obviously, **honesty goes in both ways**. Great developers should be honest enough to tell their future employers, present employers, and co-workers what they think about them, or if they believe themselves or the latter are doing something wrong or exceptionally well. Honesty is both liberating, and makes sure one avoids problems and problematic patterns.

Let Them Grow

Here's another insight: let your developers grow. If you know they have potential, then hire them regardless if they have the appropriate experience you're looking for. **No-one is born** a kernel developer, an experienced mod_perl developer, a Java developer with 5 years of experience, or a C/C++ systems programmer. But many people who are **fresh out of college or even high school**, and who like to program for fun, are capable of becoming that by being given an appropriate chance.

On the IRC, I've been talking to someone who graduated in Electrical Engineering from a British university. Since he doesn't want to work for the defence industry, he became a system administrator at a

high school. Now he knows Perl and uses it extensively, but he doesn't think he can get a more lucrative job as a Perl programmer (of which there's a lot of demand for in England) because he doesn't have a lot of mod_perl experience.

The reason people want experience is not because the people with experience are more productive, but because they know how to handle problems they encounter better [http://www.joelonsoftware.com/articles/LordPalmerston.html]. I don't claim experience is not important. However, if you're using a platform which is both reliable and predictable ("it just works"), give your programmers access to the Internet (with the Web, search engines, and IRC), to good searchable books, and to fellow developers who are more experienced than them in that platform, you can make sure they overcome such problems easily.

A different programmer, also from the UK, testified that while he started working with Perl and liked it and was good, no one was willing to give him a chance to grow [http://www.perl.org.il/pipermail/perl/2006-June/007956.html]. What is important about great hackers is not their knowledge or experience but their potential, attitude and autodidacticism. Given proper conditions they will accumulate a lot of knowledge, and surpass "medium-level" techs with a small amount of experience.

So when hiring, don't ask them highly specialised questions. Otherwise, not only you won't find too many "experienced developers", but you will also reject many great developers, who will be excellent for what you do. Training a great hacker is cheap. But hiring an experienced mediocre programmer, is a bad idea.

Take Some Good Advice

Here is some good advice I found about software management:

1. Eric S. Raymond's *The Cathedral and the Bazaar* series [http://www.catb.org/~esr/writings/cathedral-bazaar/]. Also see his "How to Become a Hacker" document [http://www.catb.org/~esr/faqs/hacker-howto.html]. Concentrates on open-source hacking.

2. *Joel on Software* [http://www.joelonsoftware.com/] - a collection of articles and essays and a weblog by Joel Spolsky. Concentrates on writing commercial, marketplace software.

3. Paul Graham [http://www.paulgraham.com/] - many essays on different topics including Lisp, language design, dynamic languages, startups, and general philosophy.

4. Extreme Programming [http://www.extremeprogramming.org/] - a software management methodology intended primarily for developing in-house software, but with some good, general ideas.

After reading all of those you'll become much more clueful about good software management than without it. Reading them taught me a lot, and even if I disagreed with some of the opinions they voiced, they were still good food for thought.

Respect and Cherish Your Developers

The final advice I can give you is to respect and cherish your developers. If you **lose one great developer**, you'll have a very hard time finding an adequate replacement for him, if you ever can. So **listen carefully to what they say**, and instruct them to tell you everything that bothers them. Make sure they are working on exciting tasks most of the time, and that they are happy.

Don't over-work them, give them the proper tools and equipment, treat them like they were kings (rather than slaves), and make sure they are happy working for you. While it does not guarantee that they won't leave you, it will probably prevent most premature quitting or getting fired.

Make sure your developers are the people who ultimately dictate what is possible and what isn't. At a previous workplace of mine, I was instructed to re-implement a PHP (and Flash 8) application in Perl. At

first I thought it was some legacy code, but as it turned out it was done because the marketing department decided that our programs should run on either PHP, Perl or ASP. However, it is common knowledge that maintaining three different codebases in three different languages is close to impossible. (The only real solution is to have a compiler from one common language into the three languages, but I wasn't instructed to do that.)

If the marketing department had understood what PHP, Perl and ASP were all about, then they would have known that writing it in PHP was enough. But there wasn't a feedback from the developers back to the marketing department.

If your developer wants to work half-time - give it to him. If he wants to work from home, or have his own office instead of working from home - give it to him. Great developers are a scarce resource, and you can't afford to lose them.

Conclusion

If you implement all of these suggestions, you'll become known as a workplace where great developers love to work, and which they will recommend to their friends. In this day and age, great developers can easily start their own companies, become freelancers, or simply remain "unemployed" while doing paid-for gigs. Or they can find a better workplace, perhaps with lower pay, but one where they will be happier.

If you're a good developer - know your worth. Refer potential employers to this essay, tell them they have a slim chance of finding someone as good as you on the market, and that you deserve the best.

Happy Hacking!