Smart and Gets Things Done: Joel Spolsky's Concise Guide to Finding the Best Technical Talent

Joel Spolsky

Smart and Gets Things Done: Joel Spolsky's Concise Guide to Finding the Best Technical Talent

Joel Spolsky

Smart and Gets Things Done: Joel Spolsky's Concise Guide to Finding the Best Technical Talent Joel Spolsky
A guide to attracting, recruiting, interviewing, and hiring the best technical talent.

A comprehensive system for hiring top-notch technical employees Packed with useful information and specific advice written in a breezy, humorous style Learn how to find great people--and get them to work for you--in an afternoon!
The top software developers are ten times more productive than average developers. *Ten times*. You can't afford *not* to hire them. But if you haven't been reading **Joel Spolsky**'s books or blog, you probably don't know how to find them and make them want to work for *you*.

In this brief book, Joel reveals all his secrets--from his years at Microsoft, and as the co-founder of Fog Creek Software--for recruiting the best developers in the world.

If you've ever wondered what you should be looking for in a resume, if you've ever struggled to decide whether to hire someone at the end of an interview, or if you're wondering why you can't find great programmers, stop everything and *read this book*.

Smart and Gets Things Done: Joel Spolsky's Concise Guide to Finding the Best Technical Talent Details

Date	:	Published May 1st 2007 by Apress (first published January 1st 2007)
ISBN	:	9781590598382
Author	:	Joel Spolsky
Format	:	Paperback 182 pages
Genre	:	Business, Management, Nonfiction, Computer Science, Programming, Science, Technology, Software

From Reader Review Smart and Gets Things Done: Joel Spolsky's Concise Guide to Finding the Best Technical Talent for online ebook

Gerd says

Good advice doesn't get old. While some of the mentioned technologies, tools, etc. are obviously outdated by now, the overall premise hasn't changed. It actually has gotten a lot harder to find good (software) developers.

I can still recommend all of the ideas in this book - tailor it to your business, location and customers, of course.

?ukasz Woli?ski says

The one thing that this book taught me for sure is that I wouldn't ever get a job at Fog Creek (author's company) ;) The book is from 2007, which means it is pretty old (as for software development world standards) and some of the advice is deprecated. However, I found most of the stuff at least enjoyable and informative.

It is probably one single book that virtually every HR person and manager that would like to hire developers should read (mostly because it is a really quick read and provides actual real world advice). Author has a bright style and made the book quite funny, so I guess it was time well spent. Also the best piece of advice from this book is in the title. The 'and' part of it is crucial. ;)

Kristiyan Bonev says

My expectations were quite different for that book. When I read "Smart and gets things done", I was like "Cool, I will learn a way to judge which people are smart and will work hard". Nothing of the sort. The book shows you the most extensive practices to hire the best 0.00001% of programmers.

Yeah, thanks for that. I mean, it's good to know and strive for it, but unfortunately not a lot of companies can do that, just because it's not always up to the employees.

It's still a good book, but probably the title has a context that one is not necessarily aware of.

Gerrit G. says

This guy knows his shit. Some parts are outdated though. Chuzpe included!

Sergey Teplyakov says

???????, ????????, ?????? ?????? ?????? ??? ?? ??? ???????? ???????????. ?? ????????? ?????, ???? ??? ?? ??????? ?????.

Inspiratielozer says

Good read but quite some information not relevant to finding the best technical talent

---

Pagalpieasian says

It sounded more like an advertisement of FogCreek. However, it did have some interesting highlights for managers and recruiters.
I wonder the author used to hire such smart people and still FogCreek end up developing products like FogBugz :/

Dana Robinson says

A great read with helpful advice for anyone who needs to hire or manage software engineers. The only real downside to the book is that, to a large extent, it assumes you make cool stuff, your company strongly supports the software developers, and that you already have at least a few rock stars at your organization. If you make boring stuff and have a shop full of average coders who are second-class citizens in your organization, there's probably not a lot for you in this book.

Victor says

Lots of great hiring and management tips, with easy to understand/read stories. I've learnt a lot from this book. There's not just ideas, there are ways that you can implement them. Definitely would be much easier to change a small organization, but they do talk about how you can try to push for change even in a bigger organization.

This book will definitely shape how I would approach hiring and managing a team.

Richard says

This book proposed that if you have the Best Working Conditions => you get the Best Programmers => to develop the Best Software => which results in Profit!

The preface for this is the the quality of the work and the amount of time spent are simply uncorrelated.

Productivity is 5 to 1 or 10 to 1 between programmers. You can't afford to be number two, or to have a "good enough" product. It has to be remarkably good, by which I mean so good that people remark about it. Having really, really, really talented software developers is your only hope for remarkableness. The great software developers, indeed, the best people in every field, are quite simply never on the market. The average great software developer will apply for, total, maybe, four jobs in their entire career. Whereas bad people are on the market quite a lot.

How to find people who are not on the market:
1. Go to the mountain
What conferences do they go to? Top end conferences or up and coming technologies
Where do they live?
What organizations do they belong to?
Which websites do they read?
Avoid advertising on general-purpose, large job boards as the bad people who are all over the market will apply and swamp you.
2. Internships
Students are lazy, with lots of options so can roll out of uni into a job. For the good ones try to attract them a year or two early - they might need some training but it is beneficial for both sides. You will likely need to have a contact at the Uni to find the best students.
If they are great make them a good offer for after graduation
3. Build your own community
Referalls
Tend to be from former companies tent do be from the same company which can be risky
Nobody wants to persuade their friends to apply for a job at their company only to get rejected
If you pay too much for referrals then they will coach people through the interview process

Work space
Private offices make programmers more productive and programmers prefer it
Putting on headphones with music to drown out the ambient noise reduces the ability of programmers to have useful insights
Office location
Does the office look exciting?
Good chairs don't cost that much more over their lifetime and if you take the cost per week it is cheaper than most other office facilities
People want to work with good, cheerful and happy people - Smart, and Gets Things Done and not a jerk
Managers can advise but they must be extremely careful to avoid having their "advice" interpreted as a command

Thing which annoy programmers
being told to use a certain programming language
people being promoted because of their ability to network rather than being promoted strictly on merit
being forced to do something that is technically inferior because someone higher than them in the organization, or someone better-connected, insists on it.

People want to work on something cool, exciting new languages attract people
Young programmers, especially, are attracted to ideological companies
open source or the free software movement
social causes
benefiting society

Developers don't really care about money unless you're screwing up on the other things - it means people aren't really loving their job. If potential new hires just won't back down on their demands for outlandish salaries, you're probably dealing with a case of people who are thinking, "Well, if it's going to have to suck to go to work, at least I should be getting paid well.". That doesn't mean you can underpay people, because they do care about justice - you do have to pay competitively, as long as the salaries are basically fair they will be surprisingly low on their list of considerations. Offering high salaries is a surprisingly ineffective tool in overcoming problems

Resumes filtering
Be selective about how we advertise jobs to limit the amount of poor CVs
Use a strictly objective system of reviewing and sorting them, this is not a filtering criteria it is just to sort a big pile of CVs to find candidates who are most likely to be suitable so they get interviewed first
Passion
Jobs with computers or experience programming going back to a very early age
People who love programming often work on their own programming projects (or contribute to an open source project) in their spare time.
Sometimes certain programming languages or technologies indicate evidence of someone who loves to explore new technologies
Pickiness
Specific covering letter to the company, a custom cover letter is a sign that if we do make this candidate an offer they're likely to accept it
programmers who can communicate their ideas clearly - so neat, well structured and gramatically correct CVs
Brains
Math camp, programming competitons etc
Selectivity
Have they been through a rigorous review process before either for Uni or another company
Hard-core
Some development work is just harder than others, if they have the harder work then they stand out.
Diversity
Trying to bring new ideas into the team - to break people out of group-think and their own echo chamber
Great developers are likely to have enough options of places to work that any extra hoops will put them off bothering to apply.
Any technology you know right now might be out of date in a year, you are looking for people who pick things up quickly and can learn new things - so don't filter CVs on key words.

Phone Interview
Get the candidate to describe their career history and basically tell me about themselves. Looking for:
Technology: How did they do things. What was their role. CV validation
Politics: How the candidate handles challenges. Looking for people who got things done, even in the face of opposition. I'm looking for people who challenged the status quo, who overcame objections, and who made things happen. Whose idea was it? Who convinced whom? Who did what? Did it work out? Why not?
Get the candidate to solve a technical problem. This should take something the candidate is familiar with but are unlikely to have implemented themselves. The aim is to look at their approach rather than getting them to speak code over the phone.
Get the candidate to ask questions about the company. This shows if they have done any research and what they are interested in.

Interviewing

6 interviewers, at least 5 peers not managers
If two people would reject the candidate end the interview at that point
Don't interview multiple people at once
There are three catorgories
Nos
Maybes - never hire maybes
Superstars
"Hire, but not for my team." is a no hire
"I'm a little concerned about" is a no hire
"Perhaps" is a no hire
It is much much better to reject a good candidate than hire a bad one
Look for people who are **Smart, and Get things done**.
Bad interviwers
Interviewers who just talk the entire time
People who are just looking for trivia e.g. "What's the difference between varchar and varchar2 in Oracle 8i?", smart does not mean knows trivia, aptitude is more important. Any skill set will be out of date in a couple of years
Good practice
Know as little as you can about the candidate in advance so it does not bias your opinion.
don't listen to recruiters opinions, don't ask around about the person before you interview them, never talk to the other interviewers about the candidate until you've both made your decisions independently. This provides the least amount of bias for or against the candidate.
Good candidates
are passionate, they might be passionate in favor or against but passion is key. Bad candidates just don't care.
can explain what they have done in a way a normal
look for signs of leadership, how have they pushed forward to get things done
write code and discuss it
pointers
recursion
data structures
ask them to find bugs in their code, even in the unlikely event there are none, to see how they approach it
Even if they are a bad candidate, you want them to like your company and go away with a positive impression.
Don't ask questions such as are they married, have kids etc even in a conversational way as this adds nothing and the candidate might feel this has been used against them which is likely illegal.
"Back of the envelope questions" e.g. How many piano tuners are there etc are a good way to provoke a conversation.
Do feedback instantly before you forget about the candidate
If 4 or 5 people think this person is worth hiring then you likely won't go wrong
If you do have to say no to someone, do it quickly and respectfully
Great people are much, much more valuable than average people - three to ten times as productive, costing 20% or 30% more

Teams
Why don't they work?
performance measurements and incentives - devastatingly ineffective
Remove the parts which are not working.
Anonymous peer ranking with the options:
Great developer

Needs specific improvements
Hopeless
Firing poor performers can increase moral because poor performers are taking time away from the good performers. If you can't fire them move under-performers to a place where they can't cause any impact.
Putting in things which do work
Three approaches to leadership
The Command and Control Method
Tell people what to do and tell them off if they don't do it
Disadvantages for developers
Smart people rebel against doing what they are told without good reasoning
Micromanaging would require a huge amount of managers to micromanage everything. That or you hit and run not seeing the consequences of your decisions.
The management have the least knowledge so are ill placed to make decisions.
The Econ 101 Method
Give them financial rewards and punishments to create incentives, aka replaces intrinsic motivation with extrinsic motivation.
When you stop paying the bonus, or when they decide they don't care that much about the money, they no longer think that they care, even though they might have cared before you started giving them a bonus for it.
They'll find some way to optimize for the specific thing you're paying them, without actually achieving the thing you really want.
You're encouraging developers to game the system.
You can't abdicate your responsibility to train your people by bribing them.
The Identity Method
Make people identify with the goals you're trying to achieve
The Identity Method is a way to create intrinsic motivation.
Make a point of of eating lunch with my coworkers. It's hard to understate what a big impact this has on making the company feel like a family, in the good way.
by sharing information people will do the right thing

Brad Pickler says

Joel is one of the best guys on the internet if you want to have read about software business, and this book follow the same quality.

Things change fast in software. The book was wrote before the Continuous Delivery got on the mouth of everybody, so the author tells about CI practices without mentioning it. I'm not criticizing it, contrariwise, I believe the author have merits in elaborating practices before all the movement start.

Yevgeniy Brikman says

This is a very quick read on how to hire programmers. It's full of insights and interesting thoughts from someone who has been in the trenches of being a programmer and hiring programmers for years, who has succeeded at both tasks, and who has thought deeply about why. He has great points on how to find programmers (hint: job boards don't work) and how to build an environment where programmers can be productive. For those reasons, it's worth reading.

However, while I respect Spolsky and have followed his blog for years, I don't agree with a number of key points in the book:

* Spolsky makes most of his arguments about hiring as if they are scientific facts, whereas most of what he says actually consists of anecdotes, correlations, and guesses. For example, when he makes the claim that interview questions about pointers can be used to distinguish between good programmers and great programmers, he has nothing but anecdotal evidence to back that up. It's entirely possible that the programmers he rejected who failed his pointers interview question would've actually been great employees. Without a controlled experiment, we don't really know. Obviously, I don't expect Spolsky to be spending his time on controlled scientific experiments, but I do expect him to present his stances as conjectures rather than absolute truths. The sad truth about hiring is that we all suck at it and not acknowledging that does a lot of harm to this industry.

* As an example of the harm this "trust me, I know what I'm doing" attitude can have is Spolsky's claim that programming ability, such as understanding pointers, is innate, and cannot be taught. I call BS on that. No one is born understanding pointers. And if a large percentage of people can't learn pointers, my guess is that has more to do with the ability of the teachers than of the students. But that's just my guess and I prefer to label it as such. Spolsky presents it as a hard fact. The either you-have-it-or-you-don't, fixed-mindset is, IMO, harmful to the software industry. We need to encourage more people to take up programming rather than scaring them away because they might have been born a muggle.

* Sposky recommends white board coding. I would argue this is a horrifically ineffective way to evaluate programmers that this industry should have abandoned long ago. Working on artificial problems from CS 101 that can fit in a 45 minute slot, writing code by hand with no compiler, no syntax checking, no auto complete, no Google or StackOverflow (yes, every programmer uses these constantly while coding), no libraries, no ability to incrementally build/run the code, no quiet time to do thinking (instead, speak all your thoughts out loud because that's totally natural!), and a ridiculous pressure to prematurely optimize the shit out of a tiny piece of code is NOT my idea of an effective interview process. A book like this recommending it as a "best practice" does harm to the industry.

In short: if you're going to hire programmers, it's worth reading this book, but don't take it as gospel.

As always, some of my favorite quotes from the book:

"Duplication of software is free. That means the cost of programmers is spread out over all the copies of the software you sell. With software, you can improve quality without adding to the incremental cost of each unit sold. Essentially, design adds value faster than it adds cost."

"The real trouble with using a lot of mediocre programmers instead of a couple of good ones is that no matter how long they work, they never produce something as good as what the great programmers can produce. Five Antonio Salieris won't produce Mozart's Requiem. Ever. Not if they work for 100 years."

"It's not just a matter of "10 times more productive." It's that the "average productive" developer never hits the high notes that make great software."

"The great software developers, indeed, the best people in every field, are quite simply never on the market [...] The corollary of that rule—the rule that the great people are never on the market—is that the bad people—the seriously unqualified—are on the market quite a lot."

"When a programmer complains about "politics," they mean—very precisely—any situation in which personal considerations outweigh technical considerations. Nothing is more infuriating than when a developer is told to use a certain programming language, not the best one for the task at hand, because the boss likes it. Nothing is more maddening than when people are promoted because of their ability to network rather than being promoted strictly on merit. Nothing is more aggravating to a developer than being forced to do something that is technically inferior because someone higher than them in the organization, or someone better-connected, insists on it."

"In a high-tech company the individual contributors always have more information than the "leaders," so they are really in the best position to make decisions. When the boss wanders into an office where two developers have been arguing for two hours about the best way to compress an image, the person with the least information is the boss, so that's the last person you'd want making a technical decision."

"The military uses Command and Control because it's the only way to get 18-year-olds to charge through a minefield, not because they think it's the best management method for every situation."

GreatBahram says

I had some problem about interviewing people and also how actually other people manage their teams. This book is like a treasure and that's not very controversial. It's hard to understate such a big impact this book has on me. I like it very much and highly recommended.

Avraam Mavridis says

I am following Joel and have read other books of him, this one seems like a summary of what he wrote in the past about hiring.

- How to advertise your positions to attract talented candidates.
- How to read and value resumes
- How to do a phone screening
...

Krishna Kumar says

A little outdated and a lot of condescension. There are some useful tips, but others I feel were just wrong. There is weird stuff about how programmers talking about O(log n) is using jargon. What comes off in the book is that Spolsky is pretty opinionated about the hiring process without doing some self-assessment of whether it really makes sense.
