# THE DEVELOPER SURVIVAL GUIDE

*Practical advice to manage difficult situations as a software developer*

*V1.2*

# Table of Contents

# 10 tips to manage outsourced software projects

First a few words about the idea of offshoring. Many developers think offshoring is a threat to their job. I can tell you right now that if you can potentially be replaced by an offshore developer, then yes, you have a big problem. You are *just a developer*. Being *just a developer* can get you replaced fairly quickly, even by another local developer. To avoid being replaced you must become a **problem solver**: a problem solver whose specialty is software development.

This post is not about the potential threat of offshoring for your job, so if you are particularly concerned by that, keep reading this blog, I have few articles directly related to developer job security coming. Meanwhile, you can check existing content that covers the subject. A few years ago, Chad Fowler wrote a book called **My Job Went to India: 52 Ways to Save Your Job**, later renamed **The Passionate Programmer: Creating a Remarkable Career in Software Development (Pragmatic Life)**. I highly suggest you purchase the book as it summarizes it pretty well.

## The advantages of outsourcing

There are several advantages for local development teams in outsourcing a part of their software projects.

**You have access to talent worldwide**: given the difficulty for western countries to find skills locally, access to talent worldwide is a real asset. There are tons of great developers out there waiting to provide you the best.

**Cost controlled to the cent**: outsourced projects can be started and stopped very quickly.

**Highly scalable**: because of the availability of the skills out there, it's very easy to scale up.

**Empower your local team**: if you use outsourcing to empower your local team instead of trying to replace it, you'll get best results.

**You contribute to a better world**: a developer you hire in India is able to support a whole family alone with only one salary. If you pay him the same rate as in your own country (which I do often when I'm financially able to), a whole village can be supported.

The only potential disadvantages are directly related to the way you manage them. Over the past 10 years I have outsourced up to 400 projects with hundreds of different providers spread over all 5 continents. Here are the 10 core principles that allowed me to reach almost 100% success rate with my recent outsourced projects. These 10 principles are divided into 3 categories: choice, supervision & strategy.

## Choice

Choice is about picking the right guys. Choice is very important and you should spend time on it.

### 1. Avoid lowest & highest bidders

The potential problem you may encounter with lowest bidders is obvious. When asked what he thought about as he sat atop the Redstone rocket, waiting for liftoff, Alan Shepard answered "*The fact every part of this ship was built by the low bidder*". What about the highest ones? It seems that

there is some correlation between bid amount and quality of the deliverable you will get. However, I found that in most cases, quality was high enough with average bidders and therefore selecting highest bidders would be a total waste of your project budget.

**2. Check ratings**

Offshoring platforms often give you the rating of previously completed jobs. That information should be taken seriously and used to make your choice. You wouldn't hire anyone without checking with his 2 or 3 previous bosses right? On a rating scale of 10, I usually discard any bid with a rating under 9 as well as bidders without any rating yet. Sometimes, I forget this rule for very small project so I can help new providers to get their first rating.

**3. Prioritize motivation**

Many bidders bid without reading your specifications carefully. I know this because I have posted absurd or incomplete specifications in the past (by mistake) and very few bidders actually reported it. Now I read all cover letters carefully and give more credit to those who actually write about the project itself. Some bidders will write about possible solutions to your problem and others will actually discuss how they will implement it. In practice, I have observed that the most motivated ones tend to perform better than the others.

## Supervision

**4. Protect your intellectual property**

Seems obvious right? But still, that's one of the most common mistakes as most of us completely forget it. Things can go bad and the company you hired for the project can claim full ownership of their work. Or worse, they decide to use what you paid for in their own businesses. Ensure that a proper NDA and intellectual property rights assignment is signed. Most offshoring platforms provide that by default, but if you plan to go alone with no support, this is something you must handle yourself and require before starting work.

**5. Refuse custom frameworks**

Very often the developers or company you hire decide to use a custom framework or library they have written. Verify it is acceptable to you. Sometimes the development shop will give you full rights on the code they wrote specifically for you but not for their libraries. This is problematic. You have a huge problem if you are so dependent on them that changing trivial things in the application requires you to hire the same development shop. It's not only about legal stuff, but also the potential complexity of the code that they wrote that makes it impossible for the standard developer to understand. Keeping the potential to continue your project without them is a really important choice that you will want to keep.

**6. Impose standards**

Make sure that they use standards in the technology of choice. Even if they don't use specific custom libraries (see previous point), you may face another problem: a specific way of coding that doesn't meet industry standards and globally accepted guidelines. In the worst case, you could be forced to rewrite everything from scratch to make any maintenance possible. I tend to give priority

to main stream technologies for that reason. PHP or .NET for instance, depending on the project I outsource.

### 7. Review early, review often

You don't want to discover at the end of the project the code did not meet your quality standards, eg missing comments, missing or poor documentation, poor coding practices, etc. Reviewing the work frequently will allow you to give feedback early in the development phase. Reviewing frequently is also the most effective way to adjust any misunderstandings of your specifications. Each review gives you the opportunity to clarify requirements.

In addition to requesting frequent demonstrations, require access to the repository. If this is not possible, request that you are sent the full source every week for review.

## Strategy

### 8. Test providers with small projects

Before I give a bigger project to a provider, I test them with one or two smaller ones. I also try to give specific projects to specific providers that have performed well in the past on similar stuff. For example, I outsourced few web design jobs to a provider through a well known offshoring platform and now I work with him directly by email because I know he will perform really well.

### 9. Accept multiple bidders to reduce risk

For critical projects, I select two or three bidders then I take the best implementation. This works best with very small projects (under $5000). It takes more of your time but it is sometimes required. It was with this approach that I discovered that bidders with motivated content perform better than others.

For bigger projects, you can use a combination of the point 8 and this one: you test multiple bidders with a small chunk of your big project and see which one performs the best.

### 10. Assemble components

Another way to reduce risk is to outsource components that your core team can assemble locally. One advantage of this approach is that you can easily switch between providers and no one really gets access to the whole thing (reduce intellectual property risks). But the most underrated benefit of this approach is that your architects are then obliged to develop an open architecture that will help you extend the application or replace whole parts of it painlessly in the future.

## Final thoughts

Applying these principles alone is not enough. You will need to fail a few times to really learn what can be written in a simple blog post. Experience gained by practice is the only way to success. Don't get discouraged if your first 2 offshoring projects fail. Offshoring empowers thousands of teams worldwide and you can do it too.

# How to convince your boss

When you need to convince your boss about something, such as upgrading your development machine, the most common mistake you can make is to talk about your problem. You'll face some situations in which your opinion, your problem, your pain is not really that important compared to all the other issues your boss must deal with. In fact you should never make the assumption that the person you will talk with is going to put your interests before his own. Of course you know many people that will be genuinely interested in your well being, but most of the time, convincing someone to do something you think is good can be very difficult, especially for the introverts. Coming up with a solution instead of the problem alone is not enough. You must learn to focus on other people's interests and come armed with facts rather than opinions.

## Focus on him, not yourself

Focus your argument on your boss's interests, and avoid mentioning yours. Let's take the example of the machine upgrade: you want a new machine because your existing one is too slow and you really dislike working on it. Don't go to your boss and tell him: "I need to upgrade my machine because mine is too slow; it is painful to develop on it". If your boss cares less about your comfort than his budget, you will certainly get answers like "Maybe later" or "I'm out of budget" or even "But you are not blocked right?". Instead, put yourself in his shoes and try to identify what's important for him.

To help you with that, try to ask yourself how he is evaluated by his own bosses (or customers). How does he report his own performance? In that particular scenario, there are good chances that he will be praised if his team brings new functionality on time and blamed in the case of late deliveries. Your lack of productivity is the pain. Try to tell him: "I need to upgrade my machine because my productivity is really badly affected right now". Be prepared to get another kind of answer! At best a formal acceptance without resistance but in many cases, a demand for clarification.

## Come with facts

Any serious boss will ask for more information to understand your needs and eventually use it to validate their decision. You will certainly meet bosses that use them to cover their own backsides. Don't wait to be asked for it, and augment your request with facts that demonstrate the real positive impact on the pain you identified earlier. Be sure to use real facts: articles written by an industry authority is not enough. There are good chances that your boss practices critical thinking and coming with blog posts from people he has never heard of will have little effect. For the example we used earlier you could calculate the startup time or the gain in compilation time. You must convert that time into a metric he is sensitive to: money. Multiply the time you save for one or two years by your hourly cost and compare it to your upgrade's costs. If the gain is real, you won't have any problem in getting your upgrade.

To increase your chances, be sure that your argument uses simple words to avoid any misunderstanding, especially if you are presenting it to non technical people. Be sure to be concise as well: you would be surprised by the number of people reading only the introduction and conclusion or who ignore any email longer than a certain size.

Formula = Boss Pain(s) * (Real) Facts * Simplicity * Conciseness

## More tips

You should never ask your boss for permission to use a given development practice. Your boss should not impose a way to work on you but should focus on the result instead. You don't tell the taxi driver how to drive his car. You don't tell the hairdresser how to cut your hair. You don't tell a coder how to work with his code. If you think your code needs refactoring. Refactor it. You are the expert; your team should decide what practices you use. Your boss is supposed to decide ***what*** to do and more importantly ***why*** you are doing it. You should take care of ***how*** you build it. If you are in the situation where your boss tells you how to work, you should ask yourself if that environment is really good for you.

If you want to attend a conference, try to compare the costs to formal training courses and be sure to put all the advantages such as the opportunity to network.

If you need a specific tool, be sure to scan the seller's website to help you find the facts to make a good case.

# Why aren't developers paid for the value they actually provide?

There are two main reasons. One is linked to the **position in the hierarchy** and the second one to the very common **personality of programmers**. You can't change the former; however, you have more control than you think on the latter.

Most developers are positioned at the very bottom of the pyramid. This is fine: usually, nobody expects anything from you other than writing lines of code. But many developers are providing much more value than the person just ahead of them, while generally having no possibility to earn more and be paid what they worth. The reason is that in our societies, we still think the salary is bound to the position in the hierarchy. The analysts or project managers are higher in the hierarchy, so they should be paid more according to the rule. The problem is easy to spot: position in the hierarchy in many cases is not directly related to the value you provide.

But why do many developers accept to be paid less than they worth? Developers and engineers in general tend to be introverted people. Introverts are often abused by extroverts. Some are aware of their value, but feel they have no control over their destiny, while others have been told since childhood that they worth less than others. The former are convinced there is nothing they can do and the latter don't know there is something to do at all. This leads to stagnation. Many extroverts are intuitively aware of that and use it heavily in their interactions with introverts. You need to take advantage of your introvert personality to survive in this world of extrovert people.

As a developer you are responsible for your current situation. If you are really paid less than you worth on the market, you need act on it.

Finally, I want to illustrate how this perverted system can sometimes work against itself. Salary grades, for instance, can be very effective in many ways, but when poorly used and/or in strongly homogeneous companies (companies with lot of different skills involved), they can lead to very difficult situations for the employer. Below is the true story of one of my best friends.

My friend started as a programmer in a big hospital. Thanks to his hard work and dedication, he quickly became Oracle DBA, which was a critical position in a company where data is both sensitive and valuable.

The hospital worked with grades. Grades are bound to your position in the hierarchy, length of employment and diplomas. Salary increases were automatic and fixed in advance. In fact, he knew how much he would earn at every step of his career.

My friend got an offer to become DBA in another company that didn't use salary grades. By accepting the offer, his salary could be increased a lot. Because he liked and respected the hospital he worked for, he decided to talk to the boss, asking for an increase. He just asked for what he was worth on the market.

The boss refused. It was impossible because of the grades and the unions would not let that happen.

My friend left.

The hospital eventually hired an external consultant (not bound to grades) and launched the recruitment process. The consultant did not know anything about the infrastructure in place, so his learning curve was huge. The hospital lost lots of money  training him.

The hospital carried on losing.  They could not fine a qualified employee to replace my friend and are still using the external consultant at 5 times what my friend asked for.

That was almost three years ago. My friend is still at his new place and climbing the hierarchy ladder very fast doing what he loves.

The hospital is still paying 5 times more.

# How to determine your current market value

What **you are worth** as a developer is what the **target market agrees to pay** to have you on board.

What the market will pay fluctuates quickly, just like the financial markets, and can't really be generalized, so websites with indicators can't help further than giving you an indication. It may not be in sync with the value you can potentially provide because what the market agrees to pay you is generally subjective.

Knowing what you are worth on the market is an important bit of information you can get using the two proposed techniques below: one for freelancers and another one for employees and other permanent roles.

As a freelance developer, I was always asked by head hunters what was my current daily rate. At the beginning, I used to give a random number based on assumptions in the hope it would be accepted. I quickly noticed that it was always accepted... I deduced I was under market rate. I started to increase my daily rate by 50€ increments for each mission I was offered until I faced some resistance. Resistance is not a formal rebuttal by the head hunter but some sort indication that they want to negotiate. In order to prevent myself being abused by the wheeler dealer type of agents, I always stayed firm on my position refusing any reduction. The accepted daily rate was my **current market value**.

For permanent roles and employees, you can't negotiate your salary upfront without doing actual interviews. Here is the process I actually used to fine tune my method and that works very well for employees:

Update your resume & post it on the two main job advertising sites in your location or profession. You don't have to be concerned about the fact that your actual boss may find it. In most cases, it will reinforce your position. The hiring process is a painful and onerous process. If he asks you why you do it, tell him the truth.

Try to get at least 5 interviews and find out information on the proposed compensation package. Be sure to include everything: salary should not be your main focus.  Location of the company, nature of the project, work environment, alignment with your career objectives are also things you should consider.

Take an average of at least 3 offers that you get, that's your **current market value**.

This last technique can be used by freelancers to *adjust* their estimation of market value. I often noticed that the market value you calculate using the first technique is always a little bit lower that your real market value. This is due to the fact that your market value for a first 3 months contract as a freelancer is lower than what you are worth at renewal date.  The hiring process for freelancer is also very painful and pricey, so it's often very easy for a freelancer to negotiate a daily rate increase with the agent at that time.  The agent has no extra effort to do to get an additional 3 months commission, so they won't risk having no commission at all by refusing to increase your daily rate a little.

Please note that this is true when the market is good.  When the market is bad, they have more choice and therefore they can replace you easily with someone else.  Knowing when you can negotiate or not is a skill you will get with time.

# 7 telecommuting tips for developers

Working from home or a private office is probably the future of knowledge age companies. It allows you to do away with commuting completely and to work in a quiet and non- (over) interrupted environment (if you decide to). It's the opposite of the open space office. The gain in productivity can be huge. Both you and your (smart) boss are potential winners in the deal, not to mention the potential real estate saving for the latter.

Unfortunately, telecommuting is not for everyone though and this may be why it is not generalized yet despite the positive conclusions of the studies on the subject. Procrastinators may find it very difficult stay as productive as they are at the office. You will also need lot of independence and self confidence. Some employees may also suffer from isolation.

**Procrastination** and **isolation** are the two problems I will address in this post. Other problems related to telecommuting won't be debated here such as:

- security concerns,
- the fact that telecommuters are less likely to be promoted (Kreitner & Kinicki, 2009),
- potential loss of thrust by managers
- communication problems with your team

Here are the 7 things I strongly suggest you to try. I have experimented with this myself during the past 10 years. They provide some tips to help avoid both **procrastination** and **isolation**. *Disclaimer: you may adapt them to your particular case and not take those empirical & personal observations as a generalization of what to do.*

## Procrastination

**1. Use a schedule - as if you were in a formal office.**

If you don't do so, you will be tempted to work too much or too little, depending on your personality. Both are problematic long term. Having a fixed schedule will not only create some sort of rule to manage your time, but also help you beat procrastination.

One of the keys to beat procrastination is to have "starters" and avoid "retarders". If you like to do non work related stuff such as reading news or browse internet, reserve 30 minutes before and after your fixed work time for it. When you feel the need to do it while you are into your work time, remember you will be able to do it after. It usually calms down the need for it and after some time, the addiction will disappear. When you arrive at your work time schedule; close everything and start working. This can be hard the first few (ten) times, then after a while the habit will kick in.

In order to avoid the frustration of having to leave off work in a middle of something, I try not to start debugging or another task that has unpredictable time duration at the end of the afternoon.

**2. Replace commuting by physical exercises and meditation/relaxation.**

Telecommuting will free up a lot of time, and you should take it as an opportunity to take care of yourself, and certainly not work more for the same salary. Consider taking up physical exercise and

relaxation or meditation. It will improve your overall productivity and will contribute to reducing procrastination (Davis & Jones, 2007).

**3. Take regular breaks**

Since you don't have your environment anymore to remind you it's time to take a pause, use the pomodoro technique. I personally taking a break of 10 minutes every 45 minutes, but you may setup your own schedule. I use special software for that called Workrave that I warmly recommend to you.

The pause can be either doing nothing (it's what I do every other break) and free your mind or do stuff off your computer such as class papers or call a colleague. In any case, this should occur in another room, and certainly not in front of a computer. I personally do three minutes of mindfulness or simple observation of what is happening outside (simply being in the present moment). Feel free to adapt this to what works best for you.

At noon, take a full break and don't eat in front of your computer. Some of you may enjoy some cooking time while others may be very relaxed by some time listening to music.

**4. Put clear limits between work and your normal life.**

This means you have a dedicated office/room you don't use for your pleasure but only work. This is very important especially if you have kids. Your office should not be used to anything else than working. This also means:

- don't work on your laptop when you are watching a movie with your family
- avoid any professional activities during the weekend
- remove all work thinking and be 100% mentally available for your well beloved

## Isolation

**5. Go to lunch outside.**

Isolation is the other major inconvenience to combat if you are affected by it. To avoid isolation a great solution is to integrate a social lunch with others. Try to do this at least once a week. Even if not with someone else, try to go outside at noon somewhere there are other people. Why not with other telecommuters?

**6. Do co-working.**

Co-working is the new trend that involves a shared working environment for people even if they have independent activity. It feels like your office, but it's not. To make this work, the co-work area must be close to your home.

**7. Don't telecommute every work day.**

If you telecommute every work day, you will progressively become more and more disconnected from your company's culture and people. It's inevitable and it will happen. Be sure to dedicate one or two days on site. When on site, go to lunch with your colleagues.

# I love software development, but I hate doing it for a boss

Unless your boss is really a bad guy, you should reconsider the feeling. If it's just that you don't want to work for *someone*, don't work at all. Because when you have your business you work for a boss much worse than your existing one: ***yourself***. You will eventually work for many other bosses that really don't care much about your feelings: **your *customers***.

If you really love software development, if it's what you are made for in life, creating your own company may be very disappointing. Today you do what you love (developing software) for most of your time. When you are an entrepreneur, you handle many other unpleasant tasks. You have no choice but to increase your work hours to be able to produce valuable lines of code. You will have to do lot of administrative tasks, handle customers (requests, complaints, questions, ...), learn legal & accounting stuff, call people, sell your stuff, handle employees (requests, complaints, questions, ...), think about tons of things at the same time ... all of this increasing your overall anxiety and fatigue.

Here is a quick test to see if you, as a developer, are likely to succeed as an entrepreneur. Answer each question by true or false.

1. I dislike the sales part of the process and prefer to make the product

2. I'm a perfectionist

3. I won't accept to be paid less than my market value for an extended period of time

4. I really like to take care of the details

5. I really dislike being criticized by other people

6. I don't have any savings or personal investments

7. I have never been fired

8. I don't have friends or family that run their own businesses

9. I am afraid of losing all my assets

10. I'm an anxious person

If you answered "True" at least three times you should seriously reconsider creating your company.

If you really want to start your own business, consider creating a consultancy company first, and then create your products later. Many successful software companies started like this. It works because the services (much more easy to get money from) pay the bills while you develop your products. If the product fails, it is no big deal.

# How to successfully enter a new company or domain

As a consultant, I worked in various domains such as utility, telecom, aeronautics, finances, insurance, etc and I've faced the problem of not knowing very much about the domain I was entering. With time, I discovered a very simple method that works: **being actively curious**.

Request a **one to one introductory meeting** with your supervisor with the purpose of getting them to explain the business domain (the big picture). It should not last more than 4 hours (or you will get bored and your supervisor exhausted). Prepare your questions in advance. Take notes.

Ask the supervisor for a **field visit**. When I worked in the telecom industry, I asked to visit the router and transmission rooms as well as the NOC (Network Operations Center). It helped the team building the software to understand the purpose instead of blindly writing code. We knew why our stuff was useful and more importantly, who will be using it, and how.

Have frequent **lunches with suits**. More frequently, I would say, than with fellow programmers. They love to talk about their stuff and if you listen carefully, you will learn lot of amazing things. Be prepared to share some amazing stuff with them too or they will soon find you very boring.

When you have the occasion, **visit the departments**. Ex: when I had to work with a marketing manager of a big finance institution, I asked how our applications helped them and what her routine was. By actively listening to her, I was able to find out new stuff to automate and make her work life easier. I suggest you make a physical visit rather than a phone call. This means you can see her doing the thing and more importantly, using your application! What is obvious for you may not be so obvious to the rest of the world.

**Buy books on the domain** and watch Youtube videos. Read your domain's information websites as well. Give priority to higher level literature. Books that are too technical could be boring or impossible to understand in the beginning. What you need is general concepts to fuel conversations and learn more.

**Talk about it** to your friends & family often. Talking about your stuff will help you integrate the huge amount of information you must learn. Prepare an elevator pitch and be prepared to give details. Don't hesitate to write it down so it is easy to remember.

**Read all the news your company publishes** - including annual reports. In fact, the first thing to do when you are hired by a new company is consult their website and publications.

When you don't understand something, don't hesitate to **ask the guy in the company** (email preferred) that can answer your question. The worse thing that could happen is being told that (s)he has no time for that.

**Remember this**: the developer that knows the domain very well is no longer just a simple developer. You are a little more than that. And so is your market value.

# 6 advantages to using third party libraries over developing your own

You should always consider using existing software components instead of developing your own; even if you think that the latter would be much better. Here are 6 reasons why working with third party projects (open source or commercial) is usually a better choice:

- **Domain expertise:** Authors are usually experts in the domain covered by the library. This will ensure that you will get the most appropriate implementation. A good example is SharpMap. The main committers are experts in geospatial software.

**- Stability:** These libraries have the big advantage of being used by other people as well as you, and in many cases, hundreds if not many thousands of developers worldwide. Most of the early problems have already been encountered by others and fixed by authors. If they don't fix them, it's a good opportunity for you to contribute and give back to the community!

**- Knowledge:** You will learn from others' code and design. Many popular libraries are written by top notch developers and are usually great examples of good coding practices and design. You will learn by just using them.

**- Finance:** You save tons of money. The equivalent of hundreds if not thousands of man days of work for free or, at the very worst, the cost of one man day for commercial libraries.

**- Support:** Paid libraries usually come with free support from top class developers that you can contact 24h a day. Many developers of free libraries also provide that level of support. Exposing your team to these developers will be beneficial for them.

**- New features:** They will appear automatically, without effort, in your product. If you are using the reporting engine from vendor X, and vendor X releases the new feature Y, you will be able to provide the new Y feature to your customer at no cost, with very low effort. You can consider the authors of your libraries as other teams working for you, for free or for very little money!

So, assuming that you are not an expert in the domain, don't have thousands of users, have lots to learn from others, don't have tons of money, will probably need support and resources are stretched, don't reinvent the wheel -unless you plan on learning more about the wheel!

Do you think NASA would have been able to send men to the moon if they tried to build the components of their rocket themselves?

# The Agile Essentials Checklist

Here a "light" Agile Software Development checklist that I have used for many years to introduce Agile in organization. I usually introduce few items per week.

## Product Management

- A product Backlog, estimated and prioritized by a "Product Owner" is used
- A "Release Plan" exists and is known by the team
- A "Company Strategy" exists and is known by the team
- Features are divided into "User Stories"
- The "User Stories" are estimated by the whole team using "Planning Poker"

## Workflow

- The development work is divided into iterations or timeboxed "Sprints" or "Iterations"
- A "kanban" or "Information Radiator" is used
- The tasks are not assigned, the team organizes itself
- The "Velocity" of the team is known
- No outsider can interfere directly with the team during an iteration
- "Daily Meetings" take place and do not last more than 10 minutes
- A "Sprint Review" is organized and the output recorded
- A demonstration is held at the end of each iteration
- The problems are tracked and by the "Scrum Master" and/or management
- A "Burndown" graph is updated daily
- The "Code Reviews" are systematically organized

## Development Tools & Rules

- A source controller is in place
- A continuous integration build server is used and testing (unit & guidelines) takes place at each commit
- The packaging of the product is fully automated
- A (simple) bug management tool is used
- Each bug is reproduced in a single test and then corrected
- 80% of the code is covered by automated testing
- A "Solution Log" in WIKI form is used
- The "Coding Guidelines" are defined and understood by all
- A maximum of 40 work hours per week!

Please note that any numbers above can be adjusted to your reality.

# The 3 rules to quitting your job with style

Quitting your job can be very difficult.  Just the idea of announcing that you are leaving can be very stressful.  But leaving a company **properly** is also required to avoid any unwanted negative effect on your career. It can even boost it: every employer you leave is a potential sponsor.

This post will explain three rules to follow in order to avoid most of the troubles that you can encounter during the process. The outcome will depend on how you **announce** it, the **feedback** you give and the **support** you provide.

As usual, to know how to behave, you must put yourself in the other person's shoes. You may know the feeling you get when you get fired. If your career is not long enough to have that kind of experience, you may remember how you felt when a loved one announced (s)he won't continue the relationship with you. When an employee announces his departure, the employer can feel exactly the same.

So let's look at it that way and imagine how you would prefer to be fired.

## Announce it in person

You will certainly prefer to receive the letter from the hand of your boss rather than by a registered letter in the cold morning.

After you write a short resignation letter, book a meeting room with your manager. I don't suggest you put every detail in the letter as this may be used against you or others. Since it is not required, I would avoid it completely.

Prepare in advance what you are going to say. Just like your letter, this has to be short too. The discussion that will emerge from your meeting can be lengthier of course, but the announcement itself should not be too long. Also be sure to prepare the announcement with a short introduction that will soften the effect a bit.

## Give honest and personalized feedback

When you announce it, the most common question will be "why?"  You would probably appreciate some honest feedback on the reason(s) why your employer decided to end your contract. The feedback you give to your employer doesn't have to be detailed and must be oriented to you, not him.

Example: if you decided to leave because they still work with an old programming language and **you** can't do anything about it, don't tell them it's wrong, but simply say that you don't feel comfortable with it. In fact, almost everything is personal; someone else might love to work with that language. Using the old programming language isn't "wrong", just *less appropriate* for you.

Here's a few recommendations for the discussion:

- be specific
- put the emphasis on how **you** feel things, how it affected you
- avoid any judgment on persons
- don't talk about problems that can't be changed

- be sure to have a neutral tone and avoid sarcasm or anger

Your feedback doesn't have to be focused only on things that can be improved. It's a very good opportunity to say what was great. Just like the reasons for leaving, prepare a list of what you really enjoyed at the company and be sure to talk about it during the last part of the discussion. It will soften the whole discussion.

## Offer your full support

Leaving a company is not only a loss in resources, but also a source of potential problems. While your employer should have taken precautions to avoid trouble in such cases, you must ensure that you can provide the minimum support required for your former company and colleagues. Even if things didn't go very well between you, you must stay professional.  Offer your support at the very end of the discussion. Be sincere but be sure to put some limits and don't hesitate to say no if necessary.

Quitting a job is never easy and in almost all cases, feelings are hurt. It is very unlikely that you will never face this situation again in the future. You may be in the manager's position one day and everything in this post also applies to employees being fired.

## Summary:

Announce it person

Give honest & personalized feedback (oriented to you)

Offer your full support

# How to become indispensable

All social groups seek the company of people who help them reach their goals. For example a network-gaming oriented group will generally be composed of people with profiles compatible with the purpose of the group: gamers. On the other hand, people with generally incompatible profiles are rejected. This becomes a problem when someone still wants to join with an inconsistent group. The only solution for that individual is to adapt and align their goals with those of the group, or find another group compatible with their current profile.

In the case of a social group, this process is more or less unconscious. In a company, it's the same thing, but with the aggravating circumstance that you generally cannot choose. If you are not aligned with corporate goals, you will eventually pay the price. On the other hand, if you understand that the closer you get to the corporate goals, and show that you will help achieve them, you will become **indispensable**.

This type of person was recently named the **Linchpin** in the latest book by Seth Godin (which has the same name). Here is his description:

> There used to be two teams in every workplace: management and labor. Now there's a third team, the linchpins. These people figure out what to do when there's no rule book. They delight and challenge their customers and peers. They love their work, pour their best selves into it, and turn each day into a kind of art.

Linchpins are the essential building blocks of great organizations. They may not be famous but they're indispensable. And in today's world, they get the best jobs and the most freedom.

This book is really worth every penny, and if you like this article, I strongly suggest you to buy it as it is full of similar content.

In this article I will tell you about a simple and effective way to become indispensable: make sure you do not create new problems, and on the other hand that you solve a lot. **Become a problem solver**.

The trick in organizations is to identify what is "*creating a problem*" and what is "*solving a problem*". For example it is not necessarily creating or resolving bugs. It's evident that if you create many bugs, you're creating problems, even though when you solve them, you are solving problems. I will focus on another kind of behavior that can be considered, generally unconsciously, the same as the act of creating problems. Some problems that you are not directly responsible for will still be associated with you just because you report them. Or worse, you don't report them. Here are examples of behavior that will be seen by your management as problems that *you* create (or bring) while this is not necessarily the case:

- You complain to your hierarchy about your work environment.
- You indicate that your application has serious performance issues.
- You report that the components suite that you purchased is not suitable for your needs.

All these examples are *your problems* until you talk to your manager. They become the problem of the manager from the moment you bring them to him. Even if you are not directly responsible for these worries, you'll be automatically *associated* with them. The average manager has no time to deal with new problems and he will try to avoid anything related to them.

The first tip is to manage them yourself as much as possible and thus to take responsibility. You have everything to gain with this attitude. Because in this case, you do not *create* the problem, *you are really solving them*! Very few people have the ability to take responsibility for such decisions. But we can develop that talent by doing. When you face a problem, ask yourself the question "*can I myself, or with the help of my colleagues solve this problem?*"

Unfortunately in many cases this is not so easy. You will simply not be able to adopt that attitude for organizational reasons, such as hierarchical approval processes. If you can't be the decision maker, you must pass it to the management. Faced by what seems to be a great constraint, there is a very simple technique that is used by the indispensables and almost always overlooked by others: report the problem **and your recommended solutions**.

Too many people limit themselves to reporting the problem and then are inevitably associated with it. When reporting the possible *solutions* you are associated with them and then credited for what you've become: **a person who solves problems.** Someone absolutely essential to keep in the organization in contrast to those who create the problems and who it would be better to get rid of. The more you act this way, the bigger your circle of influence grows. Your opinion will have more and more value. While you are improving your company, you'll also improve your own working conditions.

Just like the basic concept, the technique is extremely simple to remember. When you encounter a problem that affects you or your company, do the following:

1. Identify *possible solutions* to the problem.
2. If you think you can solve the problem directly, without the advice or approval of a tier, *do it without hesitation*.
3. You cannot (yet) make the final decision? Then *report the problem with the solutions* obtained in step 1.

Examples repeating the above examples:

**You complain to your hierarchy about your work environment**: if the problem can be solved with a purchase, specify the references of the desired material as well as all associated costs. Don't forget to explain how you are affected and how it impacts the company.

**You indicate that your application has serious performance issues:** don't excuse yourself, they don't care. Recommend potential architecture changes instead or at least an external expert that could come up with more advice.

**You report that the components suite that you purchased is not suitable for your needs**: propose a few alternatives that you have tested and always give your personal opinion (what you would pick if you were the boss).

Additional tips to write an effective problem/solution report:

- Be convincing by using an appropriate response structure (see this article).
- Write your email so your boss just has to write the word **yes** or **no** in his reply.
- If you have many problems to report together with your solutions, send them in one mail only, even if it's more logical to have separate threads in case of discussions. The average boss will generally try to avoid discussions.
- Focus on the problem and the solution and only that. Avoid talking about people when possible.

Becoming indispensable to someone is a lot about solving his problems and having the same goals. Companies are seeking problem solvers. Become one of them.

# Extra tips for software developer resume design

The design of a resume, just like its content, is very important because a well-designed resume will improve your chances of getting selected from among hundreds of others. This post will give you a few tips on how to design an effective resume. For general guidance on how to create a good resume, there are plenty online resources that will do it better than me. Here I'll give you extra tips that may help when you are specifically seeking a job as a software developer.

I believe the objective of a resume is not about getting the job. Not directly. The primary objective of the resume is to get selected from among all the others and get the interview. Getting the interview is a slightly different objective than getting the job. Getting the job offer is the objective of every interview. Then comes the final objective of getting that contract, after the negotiations. That's a different way to see things and you should design your resume to get you past all the filters between your CV submission and signing your contract.

Getting interviews, getting offers and getting the contract are three steps that will be discussed on this blog in the coming weeks. Today, let's focus on getting those interviews.

## Understanding the process for hiring a programmer

To understand why it is important to have a well-designed CV, you must understand what it is like to hire new developers for a company. When a company identifies a need or position, it is described and then published. They publish it on their website, on specialized job boards, and in many cases, and this is one of the specifics of IT world, they hire a recruitment company specialized in IT.

These professional recruiters are usually wrongly called "head hunters". Head hunters are generally involved in hiring very high profiles such as CxO. The kind of recruiter we are dealing with are processing a lot more data, talking to many more people and therefore can't use the same soft methods.

At their level, there are 2 major filters involved. One is simple computer software, usually poorly written. The other involves very busy humans: the recruiters themselves. How those recruiters work is important to know because you must create a resume that will prevent you from being ignored by their system.

First, the recruiters need to process hundreds of requests in the very vast & rich domain which is IT. Recruiters generally don't have any IT background. If they did, given the actual demand they would be working in an IT company for a better salary. They are usually people with more sales oriented profiles that are willing to learn the high level concepts of their customer's businesses. This problem, coupled with the fact they have hundreds of thousands of resumes in their database make it almost impossible to avoid the use of a search engine to do a pre-selection based on keywords. That's our first filter.

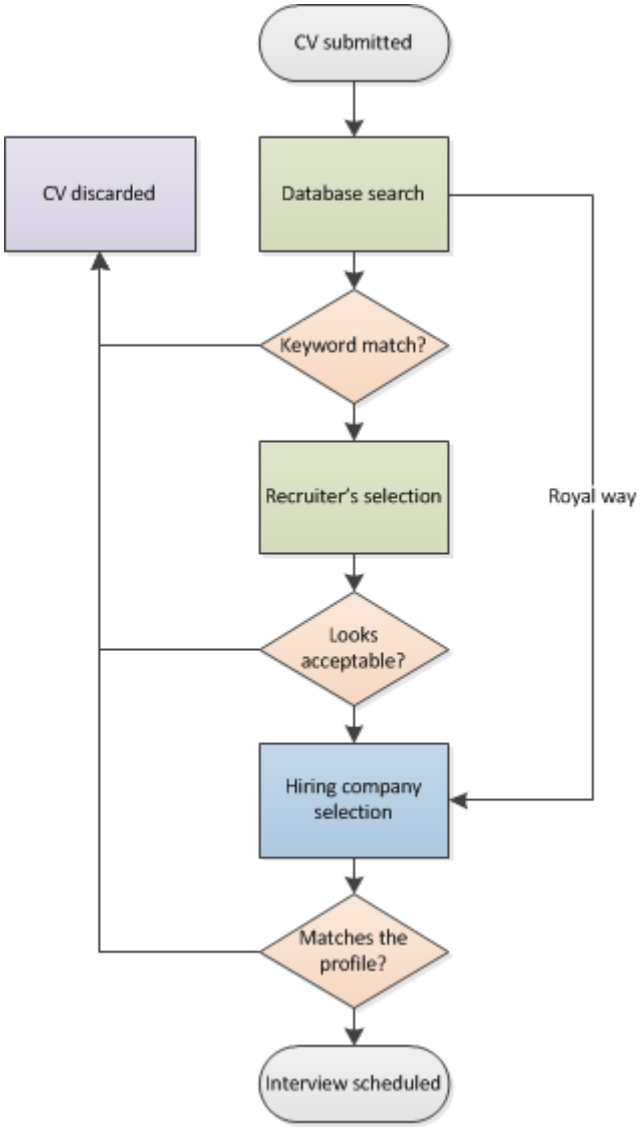
Hopefully, they don't blindly export the results and mass mail their customers. They carefully check every occurrence beforehand. Here the level of professionalism varies between recruitment companies. However they all have something in common: they are very busy. A study based on eye tracking technology determined that a resume is read in less than six seconds and they look for six main things: name, current company & title, previous company & title, previous position start and

end dates, current position start & end dates, and finally education. The rest of the data was almost completely ignored. Once your CV is picked, the final step usually involves a quick chat with you to assess your availability & confirm their understanding of your skills.

Once your resume passes that second filter, there is at least an additional one. That's the person responsible for the recruitment for the company that wants to hire someone. If you are lucky, it's the team leader of your future team, but it's not uncommon to have one or more intermediates in between depending on the size of the company. Each of them is a human filter you need to pass. In addition to everything that interests the professional recruiters, your future employer will look for additional details that will be discussed later.

When no professional recruiter is involved, you go directly to the company. It's what I call the *royal way*. The royal way is not necessarily a good thing. It usually involves an HR department you don't want to deal with and/or potentially more documents to be processed by the hiring company.

The overall system you need to pass through, simplified for the purpose of this article, looks like this:



Many of you should now understand why when you send your resume to 50 companies you receive so few responses (positive or negative).

Let's go through each filter and see what we can do about it.

## The professional recruiter

At this level, we saw that we have 2 filters: the search engine and the actual busy recruiter.

### SEO

The software is the easiest filter to pass for a developer. Despite that, I see many CVs that seem to have been designed to be stopped by them. As I said earlier, they are based on keywords and without google science. Therefore you must ensure that every single piece of technology you know well is specified on your resume. I'm still receiving offers for ColdFusion gigs despite the fact that word appears a single time on my resume and in a mission from year 2000! Yeah that makes me think, and you should be thinking too, about removing every unwanted keyword that would get CV selected for projects that you are not interested in!

Don't hesitate to specify a technology multiple times if used at different companies. As a developer we naturally aggregate that in an area called *skills*. While it would be a good idea to have such section in your resume, it doesn't help you to get enough visibility. Repeating keywords for every professional experience will put emphasis on your experience in a particular technology and make it more visible to the recruiter. If you have 5 years' experience and worked only 1 year as a java developer, it's like you have only one year of experience in that particular technology. Don't abuse those keywords. Be sure to specify tools you have actually used. I usually recommend that you put the words in bold for the technology you have mastered particularly well. On the other hand, remove any irrelevant keywords. I know you learn COBOL at school, but it's already 5 characters too much on your resume.

### One page

The number of pages in your resume doesn't matter much. What matters however is the first page. When you do a search on your favorite search engine, you rarely look at the second page. Ensure that the essential information is there **on the first page**. Here is a list of things important to the recruiter that need to be there. There are other things that are important to the hiring company that will be added in the next section of this article. Here we focus on what the recruiter will look for:

- Keep contact information short.
- Choose a good title for your resume. Common mistake is to put your education there. I usually recommend putting "Software Developer" which is the perfect description of a software developer :)
- Start with a short description of your expertise. It should not be more than 2 sentences. Don't be too precise there. You'll get into details later.
- List your professional experience first, from the newest to the latest. Be sure that the 2 latest are present on the first page. Each experience should have a start & end date, a position title, the company name, a brief description of the project (3 to 5 sentences, easy to read) and of course the list of technologies you used.

- Don't abuse titles. They are deduced by the description of your mission. They will be discussed in the interview as well.
- Put all languages you speak on the last page. If it's important for the recruiter, he will look there.
- Avoid including your picture. The average human will unconsciously judge you based on how you look and you want to leave that for the interview. A recruiter will put even more emphasis on it because he will try to anticipate the judgment of his customer.

### Gaps & job hopping

All this advice is far from being enough. You have already polished your CV to seduce professional recruiters, now you need to ensure that you don't have anything in it that will be a blocker. The recruiter is playing his reputation, so he will systematically eliminate any resume that seems too dangerous. Two things indicate you are a potential danger (even if it's not true): **gaps** and **job hopping**.

Start & end dates of your experience are very important. Gaps can have many explanations. As a coach for developers seeking for jobs, I see those gaps very often. When I ask what the candidate did in between I get all kinds of answers and surprisingly, the explanations I get from the candidates are actual extra bonus points instead of the expected negative impact. If you took one year out to start a company (even if you failed); put it in your resume. If you took one year to visit Asia with your family; put it in your resume. If you went back to college; put it in your resume. A single sentence explaining the gap is more than enough.

Gaps are not the only disqualifying thing related to dates. Job hopping can be a serious problem too. Having too much job experience in a short period of time can be seen as instability, and therefore a potential problem to deal with in the future. The hiring process, not counting the internal training every newcomer has to do, is a very costly process for the company. Therefore, it's easy to understand why every employer is looking for (very) long term collaboration. The only way to soften how this can be perceived is ensuring that you put lot of visibility on contributions you made for each company your worked for. Make it evident that hiring you will provide value, even if you don't stay there for years. Please note that while job hopping is generally seen as a bad thing for employees, the same is not necessarily true for freelancers.

### The hiring company

From multiple hundreds of matching resumes, the employer can still receive dozen of CVs on a daily basis. Believe it or not, many of them match their profile requirement. It doesn't mean every profile fits the position, it means that there is more work for the hiring company to filter them out (again, since the professional recruiter has already pre-filtered them). He doesn't want to spend the whole day reading that pile of paperwork, especially when the person recruiting is a different person than the person who is actually in need of that additional resource. Another common situation…

More importantly, nobody has the time to interview dozen of developers. So a massive filtering will occur just for the interview. Your primary objective is to pass that very strict filter and be in the first selection, just like you want to be in the first page of a google search. You want to give your prospective colleague the chance to meet you. In addition to all the tips specified above here are some that are usually overlooked.

### You solve problems, you don't create them

First if you have a technological blog, participate in developer communities, wrote a programmer's book, or all of the above, that's something you want to specify on the first page of your resume. This will give you extra bonus points and many recruiters, especially if they have an IT background, will give lot of value to that.

Secondly, the company wants to know how you will help them solve their problem. The best way to show you are the guy they need is to write a short description of the problem you solved for each experience listed. Don't limit yourself to describe what you did, but write why and how.

> I developed a new backend system for their e-commerce website using ASP.NET and SQL Server.

How boring! Instead put yourself in the shoes of the recruiter and write something like:

> Developed their new backend system allowing agents to process an increased number of orders and provided them with tools to improve communication with their customers. Facing aggressive deadlines & heavy pressure we heavily relied on extreme programming and scrum. We exploited the possibilities of ASP.NET & SQL Server to the limit.

Of course, everything you write must be true. You will have already read on the many other resources about building a great CV that telling the truth is the golden rule.

The texts above have been invented for this article, but having worked on these with dozens of developers, it's generally possible to get amazing descriptions just by thinking a bit. By extension, be sure to avoid giving unnecessary facts. By removing useless information, you free up some space to put what is relevant to your future employer.

### Miscellaneous & hobbies

Knowing what is relevant is difficult and one area that is often overlooked is miscellaneous or hobbies. Too often, it is completely removed for the purpose of a clearer CV. I believe it is a mistake because it is a great opportunity to tell them a bit more about you and influence the decision to get you into an interview. It helps the recruiter to understand how the candidate manages his life but more importantly, it gives information on his soft skills. I will take two real life examples.

The first wrote "sports, computing & movies". When you talk a bit more with him about that, you discover that he runs intensively and often competes in marathons. Marathons? Yeah, that guy knows what effort is doesn't he? I also recommended that he mentioned what kind of movies. I bet it is sci-fi ones, and I bet it would interest many other recruiting developers :)

The second one had "scouting" in the hobbies he listed. After few questions I actually discovered he was a chief scout, managing dozens of kids every weekend and in summer camps. That's someone I need in my team!

Which aspects of your personality do you need to say more about and which are important points for recruiters? Ask yourself the question "what am I doing when I'm not programming?" and put

everything you think would interest the recruiter. Don't minimize your skills. Instead, put emphasis on them, while staying as humble as possible.

## Conclusion

Your resume should demonstrate you are a problem solver and should avoid any indication that you are going to create problems. It's a document that is supposed to sell yourself so there is no shame on putting emphasis on every good thing in you.

Summary of the tips:

- The first page is the most important one. Polish it.
- Use appropriate keywords, and highlighted when appropriate.
- Fill the gaps with relevant information.
- Anything that differentiates you from the lambda developer should be on the first page.
- Write attractive experience descriptions focused on your problem solving abilities.
- Don't underestimate what you do outside programming. Use it to tell more about your capabilities and soft skills.

# How to choose a technology that has a future

As you build your career, you must make choices. One of the most difficult choices to make is in deciding which technologies to invest in. It is not unusual to see consultants who have invested years of their lives in a technology that will eventually be abandoned by the market or its designer. It is virtually impossible to predict the future at this level and the best strategy is to learn how to learn, and be a **programmer**, regardless of the technology or language. Above all it is always better to invest in things you enjoy doing. On the other hand there are simple ways to get an idea of what is in demand right now and act accordingly. Simply check for the skills that companies are currently seeking. The law of supply and demand - get it?

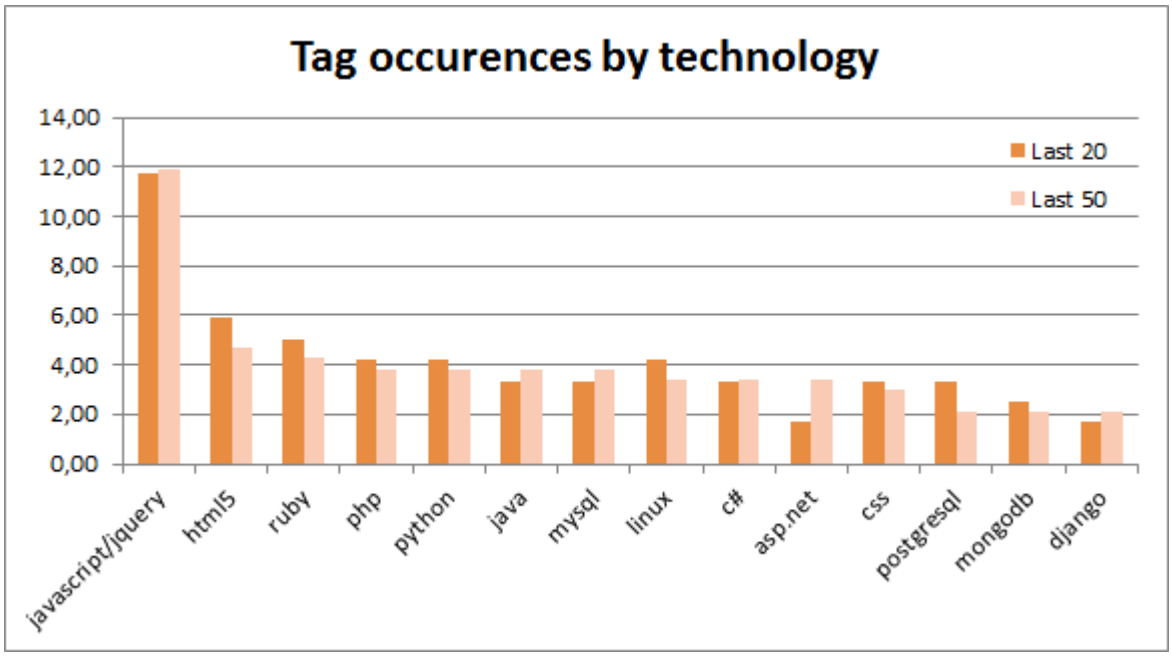
One effective method to get answers is to go on large sites offering jobs such as jobserve.com. At the time of writing this post, joberve.com offers no less than 14.626 jobs in IT, just for the United Kingdom! If you add other countries like the United States, France or Germany, the figure explodes. The ideal is to use a site that is popular in your region or country.

Browsing the latest offers of employment will give you a pretty good idea of what is being sought. The more criteria you add, the more accurate will be your answer. So if you can only work in the London area, there is no need to search in San Francisco. It is pretty obvious, the nature of IT projects developed in these two cities can be very different.

This research can be tedious. Fortunately, another site that offers jobs provides a faster way to determine the technology application, but at some cost. This is Careers 2.0 by StackOverflow and has an amazing technology tag feature.


I was able to obtain figures in under 5 minutes using the following technique: I created a sheet in my favorite spreadsheet. I typed each tag that I found, one after the other - for the latest 25 bids only (1 page) initially. I then aggregated the occurrences and found myself with a list of technologies and

the number of times each was mentioned overall. The following result is for the latest 25 or 50 job offers near London (occurrences are in % for comparison).




I compared 25 to 50 because it is well known that the more data you have, the more accurate the result you have. With the 25 latest job offers, I got a total of 120 tags, reduced to 56 after aggregation. When I took the latest 50 job offers, I got 235 tags reduced to 89 after aggregation. You can immediately see without having to do much calculation that the latest 25 job offers is more than sufficient. Adding more offers did not change the result significantly, keeping a satisfactory validity for our purpose: having some indication of what is in demand.

Now that we have our results for **Careers 2.0**, can we generalize them (external validity)?

Of course not! And this is the main issue. There is an obvious *website bias* involved. Careers 2.0 provides us with a convenient way to test technologies thanks to their amazing tagging system, but companies posting on it are not representative. It has only 83 job offers for the London area at the time of the test (12th of July 2012). Jobserve.com on the other hand has 8,657 jobs available for the same criteria.

Careers 2.0 is growing fast, so we can expect that in a few years it will be the reference for programmer's job vacancies and will provide more accurate data. For now, use them as an indication and nothing more. Until then, be sure to develop your ability to learn and give priority to technologies you love working with rather than just the ones that are currently in demand. Success almost guaranteed!

# Fast learning for ambitious programmers

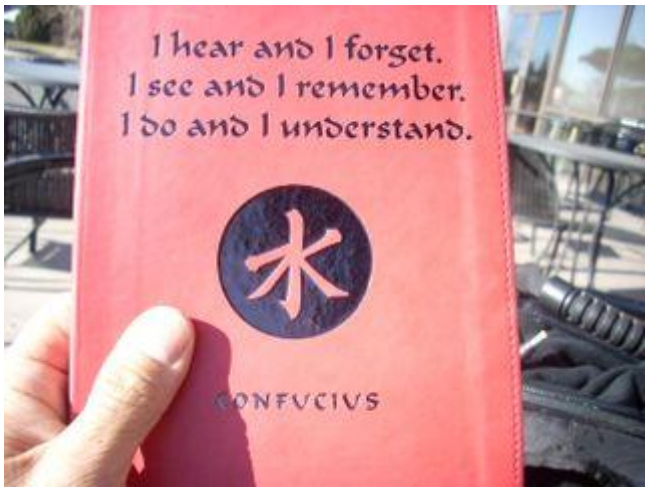
Learning has been one of my favorite activities since my childhood. I've tried many methods and found those that were the most appropriate for me. We are all different and the method that works perfectly work for George could fail with Amy.

Developing your ability to learn, before everything else, is the only viable job security strategy you could have. In fact, stopping learning is the most effective way to screw your career. Developing your ability to learn will not only help you to get more attractive jobs, but it will also increase your intellectual performance.

In this post, I'll describe a few techniques I found useful during my career in the hope you find them interesting too.

## Practice

The most obvious one you may think, but it seems that it's also the most overlooked. I see many developers reading a technical book from cover to cover, closing it, then starting a new one. This is far from being effective.




I hear and I forget,
I see and I remember,
**I do** and I understand.
[Confucius]

This is really my favorite quote because it summarizes it all. You should put everything you read into practice, when possible and when appropriate. It is how the learning process works best. Many of the things we read are not understood until put into practice. And it is very likely that you'll forget what you don't understand, or worse, apply it improperly.

Without going into details, there is a technical explanation for this. Practicing involves many other areas of your brain and it helps integrating everything. Here is some additional advice specifically related to technical books, blog posts or conferences:

- When you finish a chapter, do the exercises. All of them, even the most simple ones. In fact you should give priority to books with exercises.
- Try to summarize every chapter to a colleague. It will force you to deconstruct and therefore understand it. Talking about something you read also involves a different part of the brain and it helps with integrating the knowledge. You don't have to speak to someone: you can just simulate a discussion. It will work just as well.
- Put it together in a test project you build from scratch.
- Don't limit yourself to the single source of knowledge. Google everything you learn and read the different positions and perspectives available. Writing your own synthesis can help a lot too. Why not in a blog post?

## Get feedback. Give feedback.

As an extension of practice, seeking feedback is another great learning method. In programming, one of the most effective ways I know is pair programming & reviewing. Don't wait for it. Ask for pair reviewing or programming actively. If you are a solo developer, consider online reviewing platforms.

Also be sure to read as much code as you can. Open source projects are a great fit for that. It's how many of us learn the features of the framework or language they are using.

> Oh I wasn't aware I could do that in just one line of code! [Random learner]

Another effective way to learn is to actually teach. Teach another what you are supposed to learn. When you teach others, you are responsible for the information you give and you have special motivation. That one to many relationship is not the only available one. Participating in communities can stimulate your learning process. "Being there" makes you learn from other's knowledge & experiences. As a bonus, you learn how to teach or transmit your own knowledge.

## Overcome information filters

Our brain is constantly overloaded with environmental data and we have developed a kind of mental filter that is supposed to let pass only the information important to us. This type of filter also stops our creativity from expressing itself and blocks the acquisition of knowledge - even the knowledge that will be useful in the future.

Stanley Kubrick has developed a special technique that he describes in his biography. According to him, this technique has allowed him to acquire considerable knowledge, increase his creativity and his ability to learn. The most visible and obvious result is his films.

The technique was extremely simple: he went regularly to a library and picked up a book at random, without even looking at the title. He read to the end even if he did not like it. This technique has forced his mind to expand into new territories by preventing the mental filters from operating. The new knowledge gets integrated with the previous and creates new perspectives.

You can train yourself like that with anything, not only books. As a programmer, it is often recommended you learn a completely different language than the one you have used every day for years. Just to give you another perspective on things. In fact, too much bias towards technologies or ideas often stops you from advancing and evolving to the next level.

## Fail

If you don't fail sometimes, you are not putting enough effort into what you are doing.

Failure is the most natural learning process you can find. In fact, we are physically and mentally wired to learn from failures.

> I make more mistakes than anyone I know. And eventually I patent them. [Thomas Edison]

I do not suggest that you start doing anything carelessly. I suggest that you stop being afraid of failure. Fear of failure prevents you from doing the necessary experimentation to advance in life.

Fear of failure is also completely irrational. Since most people try to hide their failures, we only have a tiny part of the information and we think people who never fail are the norm.

Experience is how you should describe failure.

## Learn by choice

I kept the most effective way to boost your knowledge for the end. This is a very simple technique that I developed over years. In my research of novelty, I was always attracted by new technologies and new domains. I discovered that this way of behaving helped me to gain knowledge faster than choosing the path of least resistance.

Every single time I had to choose between two or more projects, I took the most difficult and challenging one. The one that contained the most unknown technologies or the one in a new domain I wasn't familiar with. Never stay in your comfort zone.

The most difficult part is not deciding to take the hardest path. The most difficult part is to justify your lack of experience in a given technology when an employer is specifically looking for it. Hopefully, you'll be able to convince them that you are a fast learner and someone will give you the opportunity to show it.

And there is a positive *side effect* with this method: job satisfaction. The combination of competence & challenge leads you to satisfaction. I will end this blog post with a link to a video related to this that is really worth watching.