Extracted from:

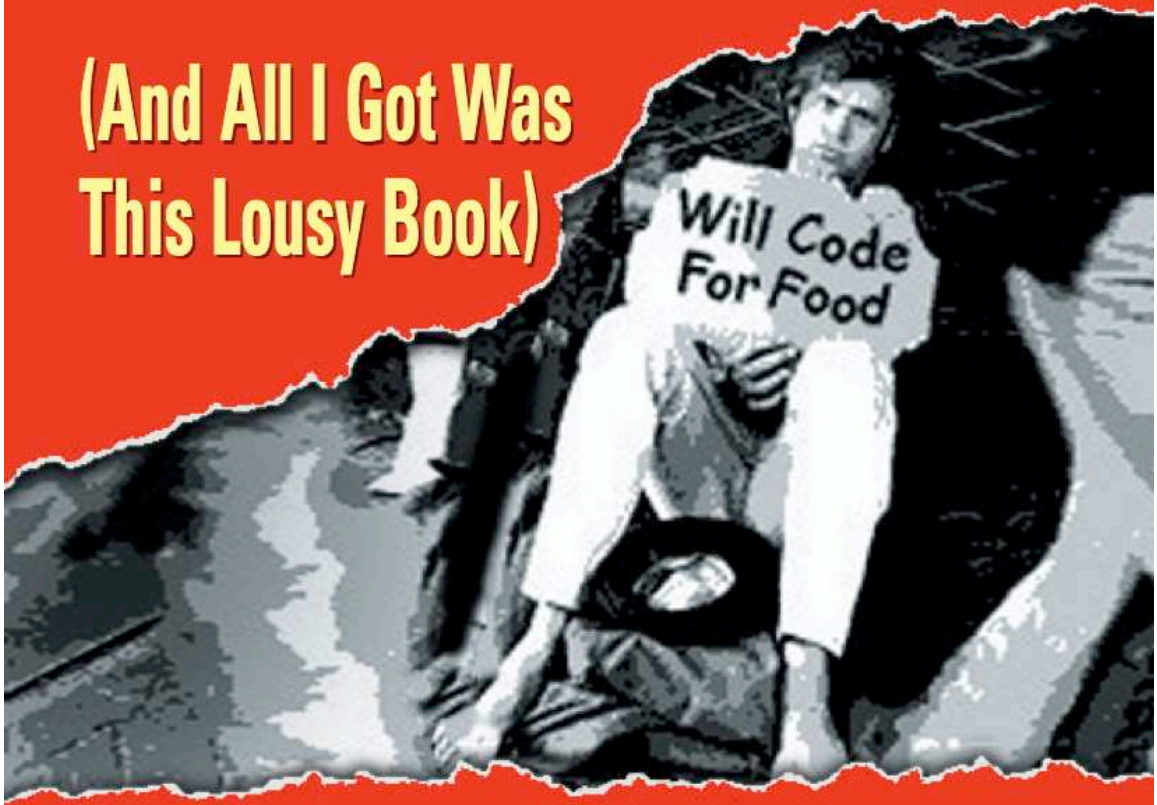
My Job Went to India

And All I Got Was This Lousy Book

The Pragmatic Programmers

MY JOB WENT TO INDIA

(And All I Got Was This Lousy Book)



52 Ways To Save Your Job

Chad Fowler

5 Be a Generalist

For at least a couple of decades, desperate managers and business owners have been pretending that software development is a manufacturing process at heart. Requirements specifications are created, and architects turn these specifications into a high-level technical vision. Designers fill out the architecture with detailed design documentation, which is handed to robot-like coders, who hold pulp-fiction novels in one hand while sleepily typing in the design's implementation with the other. Finally, Inspector 12 receives the completed code, which doesn't receive her stamp of approval unless it meets the original specifications.

It's no surprise that managers want software development to be like manufacturing. Managers *understand* how to make manufacturing work. We have decades of experience in how to build physical objects efficiently and accurately. So, applying what we've learned from manufacturing, we should be able to optimize the software development process into the well-tuned engine that our manufacturing plants have become.

In the so-called software factory, the employees are specialists. They sit at their place in the assembly line, fastening Java components together or rounding the rough edges of a Visual Basic application on their software lathes. Inspector 12 is a tester by trade. Software components move down the line, and she tests and stamps them in the same way each day. J2EE designers design J2EE applications. C++ coders code in C++. The world is very clean and compartmentalized.

Unfortunately, the manufacturing analogy doesn't work. Software is at least as malleable as software requirements. Things change in business, and businesspeople know that software is *soft* and can be changed to meet those requirements. This means architecture, designs, code, and tests must all be created and revised in a fashion more agile than the leanest manufacturing processes can provide.

In this kind of rapidly changing environment, the flexible will survive. When the pressure is on, a smart businessperson will turn to a software professional can solve the problem at hand. So, how do you become that person whose name comes up when they're looking for a superhero to save the day? The key is to be able to solve the problems that may arise.

What are those problems? That's right: you don't know. Neither do I. What I *do* know is that those problems are as diverse as deployment issues,

critical design flaws that need to be solved and quickly reimplemented, heterogenous system integration, and rapid, ad hoc report generation. Faced with a problem set as diverse as this, poor Inspector 12 would be passed over pretty quickly.

The label *jack-of-all-trades—master of none* is normally meant to be derogatory, implying that the labelee lacks the focus to really dive into a subject and master it. But, when your online shopping application is on the fritz, and you're losing orders by the hundreds as each hour passes, it's the jack-of-all-trades who not only knows how the application's code works but can also do low-level UNIX debugging of your web server processes, analyze your RDBM's configuration for potential performance bottlenecks, and check your network's router configuration for hard-to-find problems. And, more important, after finding the problem, the jack-of-all-trades can quickly make architecture and design decisions, implement code fixes, and deploy a new fixed system to production. In this scenario, the manufacturing scenario seems quaint at best and critically flawed at worst.

Another way in which the software factory breaks down is in that, although in an assembly line the work keeps coming in a steady flow, software projects are usually very cyclical. Not only is the actual flow of projects cyclical, but the work inside a project is cyclical. A coder sits on the bench while requirements are being specified, architected, and designed, or the coder multitasks across many projects. The problem with multitasking coders is that, despite the software factory's intentions, when the rubber meets the road, the coders rely a great deal on context and experience to get their jobs done. Requirements, architecture, and design documents can be a great head start, but ultimately if the programmers don't understand what the system is supposed to do, they won't be able to create a good implementation of the system.

Of course, I'm not just picking on coders here. The same is true at nearly every spot on the software assembly line. Context matters, and multitasking doesn't quite work. As a result, we have an inefficient manufacturing system. There have been various attempts to solve this problem of inefficiency without departing from the manufacturing-inspired system, but we have not yet figured out how to optimize our software factories to an acceptable level.

If you are *just* a coder or a tester or a designer or an architect, you're going to find yourself sitting idle or doing busywork during the ebbs of your business's project flow. If you are *just* a J2EE programmer or a .NET programmer or a UNIX systems programmer, you're not going to have much

to contribute when the focus of a project or a company shifts, even temporarily, out of your focus area. It's not about where you sit on the perceived value chain of project work (where the architect holds the highest spot of royalty). It's about how generally useful you make yourself.

If your goal is to be the last person standing amid rounds of layoffs and the shipment of jobs overseas, you better make yourself *generally* useful. If you're afraid that your once-crowded development office will become home to an onshore skeleton crew, it would serve you well to realize that when the team has only a few slots, a *just-a-tester* or *just-a-coder* is not going to be in demand.

Generalists are rare...and, therefore, precious.

Something I have personally experienced in searching for employees in low-cost countries is that there aren't many generalists. The Indian IT industry, for example, was formed in the image of its cultural heritage—one that places great emphasis on rank and title. I interviewed people calling themselves *team leaders* who led teams of two (self-inclusive) and reported to managers of two such teams. In many cases it can get so ridiculous, that the organizations' structures are parodies of themselves.

The software factory system of development is a perfect fit for the Indian IT sector, because it naturally supports the hierarchy that the companies and their employees desire. Testers are the bottom rung of the ladder, and nearly everyone you meet there wants to become an architect and then a high-level manager. The culture breeds specialists. Architects don't stoop to design. Designers don't stoop to code, and so on.

The way to become a generalist is to not label yourself with a specific role or technology. We can become typecast in our careers in many ways. To visualize what it means to be a generalist, it can help to dissect the IT career landscape into its various independent aspects. I can think of five, but an infinite number exists (it's all in how you personally divide topics):

- Rung on the career ladder
- Platform/OS
- Code vs. data
- Systems vs. applications
- Business vs. IT

These are different dimensions on which you can approach the problem of becoming a generalist. This is just a way to think about the whole pic-

ture of your career, and you can probably come up with a better list for yourself. For now, we'll discuss these.

If companies need generalists, they're going to have to get them in the West.

First, you can choose to either be a leader or manager type or be a technical person. Or, you might pigeon hole yourself into architect as opposed to being a programmer or tester. The ability to be flexible in the roles you can and will fill is an attribute that many people don't understand the value of. For example, while a strong leader should avoid pinch hitting as often as possible, the new world of onshore skeleton crews can benefit from a person who knows how to lead people and projects but can also roll up their sleeves and fix some last minute critical bugs while the Offshore team is sleeping. The same is true of a software architect who could perhaps dramatically speed up progress on a project if he or she would only *write some code* to get things moving. When it comes to hierarchical boundary crossing, it's most often not reluctance that stops people from doing it. It's ability. Programmer geeks can't lead and leaders can't hack. It's rare to find someone who's even decent at both.

Your skills should transcend technology platforms.

Another artificial (and inexcusable) line gets drawn around platforms or operating systems. Being a UNIX Guy who refuses to do Windows is increasingly more impractical as the jobs flow away. The same goes for .NET versus J2EE or any other such infrastructure platforms. Longevity is going to require that you are platform neutral in the work place. We all have our preferences, but you're going to have to leave your ideals at home. Master one and get good at the other. Your skills should transcend technology platform. It's just a tool. If we want a Windows person, we can hire them in the Philippines. If we want someone who really understands Windows and UNIX development and can help us integrate them together, we're probably going to be looking Onshore. Don't get passed up over what is essentially team spirit.

The dividing line between database administrator (a role that has solidified out of nothingness over the past decade) and software developer should also be fuzzy. Being a database administrator, or DBA, has in many organizations come to mean that you know how to use some GUI admin tool and you know how to setup a specific database product. You don't necessarily know much of anything about how to *use* the database. On the flip side, software developers are growing increasingly lazy and ignorant about how to work with databases. Each side feeds the other.

What first amazed me most when I entered the information technology field was that many well-educated programmers (maybe *most*) didn't know the first thing about how to set up the systems they used for development and deployment. I worked with developers who couldn't even install an operating system on a PC if you asked them to, much less set up an application server on which to deploy their applications. It's rare, and refreshing, to find a developer who truly understands the platform on which he or she is working. Applications are better and work gets done faster as a result.

Finally, as we discussed in *Coding Don't Cut It Anymore*, on page 16, the wall between The Business and IT should be torn down right now. Start learning how your business operates.

Act on it!

1. On a piece of paper or a whiteboard, list the dimensions on which you may or may not be generalizing your knowledge and abilities. For each dimension, write your specialty. For example, if *Platform and Operating System* is one of your dimensions, you might write *Windows/.NET* next to it. Now, to the right of your specialty, write one or more topics you should put into your TO-LEARN list. Continuing with the same example, you might write *Linux* and *Java* (or even Ruby or Perl).

   As soon as possible (some time this week at the latest!), find thirty minutes of time to start addressing at least one of the TO-LEARN items on your list. Don't just read about it. If possible, get some hands-on experience. If it's web technology, then download a web server package and set it up yourself. If it's a business topic, find one of your customers at work and ask them to go out for lunch for a chat.

The Pragmatic Bookshelf

The Pragmatic Starter Kit series: Three great titles, one objective. To get you up to speed with the essentials for successful project development. Keep your source under control, your bugs in check, and your process repeatable with these three concise, readable books.

Facets of Ruby series: Learn all about developing applications using the Ruby programming language, from the famous Pickaxe book to the latest books featuring Ruby On Rails.

The Pragmatic Bookshelf features books written by developers for developers. The titles continue the well-known Pragmatic Programmer style, and continue to garner awards and rave reviews. As development gets more and more difficult, the Pragmatic Programmers will be there with more titles and products to help programmers stay on top of their game.

Visit Us Online

My Job Went to India
`pragmaticprogrammer.com/titles/mjwti`
This book's home page, including errata and other resources.

Register for Updates
`pragmaticprogrammer.com/updates`
Be notified when updates and new books become available.

Join the Community
`pragmaticprogrammer.com/community`
Read our weblogs, join our online discussions, participate in our mailing list, interact with our wiki, and benefit from the experience of other Pragmatic Programmers.

New and Noteworthy
`pragmaticprogrammer.com/news`
Check out the latest pragmatic developments in the news.

Buy the Book

If you liked this PDF, perhaps you'd like to have a paper copy of the book. It's available for purchase at our store: pragmaticprogrammer.com/titles/mjwti.

Contact Us

Phone Orders:	1-800-699-PROG (+1 919 847 3884)
Online Orders:	www.pragmaticprogrammer.com/catalog
Customer Service:	`orders@pragmaticprogrammer.com`
Non-English Versions:	`translations@pragmaticprogrammer.com`
Pragmatic Teaching:	`academic@pragmaticprogrammer.com`
Author Proposals:	`proposals@pragmaticprogrammer.com`