

A n o n y m o u s C o d e M o n k e y

Your code does not start at the compiler nor does it stop at the JVM.

One lesson that most developers don't learn is to debug outside the debugger. As an engineer there are times when you need to troubleshoot, problem solve, and debug not just your software from the comforts of your favorite IDE but the whole software stack, network, hardware, user's environment, and even the user himself.

You can unit test and statically analysis software, but you can't probe your users.

Every problem, issue, and bug experienced by the end user directly and indirectly with your software eventually needs to be implicitly and explicitly dealt with by your software development team.

I wish development teams spend more time streamlining their process rather than prematurely optimizing their code.

It is OK to have software with bugs, bugs can be fixed. It is not OK to have software with excuses!

Learn, plan, design, code, integrate, build, release, rinse, and repeat.

Management has a way of over emphasizing the blatantly obvious.

I'm a lazy loading type of programmer.

I've discovered that Windows security is an oxymoron!

I know that techies, myself included, are always bragging about how their code is poetry but I have never meet a poet laureate in a development team.

Remixing and mashing up Google Maps and Flickr is like the 'Hello, World' first program of Web 2.0 mashups.



Open Source code equals community.

As a rule of thumb, I never cache, pool, or use as singleton mutable objects.

Bugs by nature are out of the box, as a developer, you need to expand the box.

WWGD: What would Google do?

At some companies, the term Spec stands for Speculation.

I went to school to learn how to program software applications, which inevitably have bug defects. There was no course at my university on testing, debugging, profiling, or optimization. These things you have to learn on your own, usually in a tight deadline.

To most Java developers, Ruby/Rails is like a mistress. Ruby/Rails is young, new, and exciting; but eventually we go back to old faithful, dependable, and employable Java with some new tricks and idioms and we are the better programmer for it.

You might as well hire your customers and pay them 50K/year because they are your QA team.

There is a saying, those who can, do; those who can't, teach. It can be said that in software engineering, those who can, code, those who can't, manage.

The greatest thing about Ruby on Rails is neither Ruby nor Rails, the best aspect of Rails is that it questioned the 'best practices' (and worst nightmares) of the current state of web

development with its philosophy of Convention over Configuration and Don't Repeat Yourself principle.

There is more than one way to do it, but do it however more than one developer can understand it.

Design a software solution for the end user, not for a fellow developer, or worse yet your database.

If the end user can't use your software, that's a bug.

You may consider your code as art but no one is going to hang your code in the Louvre.

Use the quicksort approach to problem solving.

Having code working on the developer's machine is not a valid solution, software needs to work at the client's site too!

The anti-FUD: Fighting FUD with more FUD

Unfortunately, there is no performance enhancement supplement you can take to make yourself a better software engineer.

Even the best process and best technology fail, if the people fail. People often fail, when management fails to provide the right incentive, motivation, and passion-oriented environment that people like to be a part of.

Failure is a form of experience.

Everyone has their own way of learning and should be granted the opportunity to digest information as they know how to learn best, whether it is through documentation, screen shots, class diagrams, use cases, prototyping, refactoring, and even rewriting.

It is commonly said that you don't know what you have until you lose it. With software you usually don't know what the user will want until you ship it.

As a developer, it is not like we get royalties on our work, so what do I care that 5% of my software is pirated, those that do pay for it make up the perceived loss.

Software companies should listen to their customers, even those that use their software a few times, instead of their lawyers that most likely never have used their software.

Every piece of code has bugs, but not every bug needs to be in the code.

Testing is not a one time, one shot deal. Testing is like marriage, you have to work at it every day for it to be successful.

I don't hire software engineers to code. The primary function of an software engineer is to learn and analysis and decode a problem mentally so that then can later code the solution in their language of choice.

Software engineering is nothing like civil engineering. Software engineers listen too much to their end users. If civil engineers listen to their users as much we would software developers do most buildings would end up like the Winchester Mystery House with a freeway offramp coming right up to their driveway.

This feature might have been due yesterday, but telling me a day after it is due is not a schedule!

A business laptop given to a knowledge worker should be treated such as a marine treats his government issued assault weapon, with the utmost respect and intimate knowledge of its inner workings. A business laptop is usually loaded with sensitive and proprietary information, and it is just pathetic when someone can't change the display on their own machine to use a projector.

If you don't understand Open Source licensing, don't start an Open Source project. Keep your code!

The real value of code does not lie in the source code, the value lies in the knowledge and expertise of the community.

For every one line of code in jQuery, there is at least one plugin written by a third party community member.

There cannot be a killer app without a killer community.

The easier it is to reproduce a bug, the easier it is to fix.

You can't have five top priorities at once.

As a software engineer, it is better to think of yourself as ninja programmer rather than a mercenary coder.

It's not a team meeting if only one person talks, that's more like a presentation.

The anatomy of a Google application is sheer simplicity, great user interface and experience, accessible data API, and instant scalability.

A properly written application might scale to millions of users, a top developer at most can scale to a few tasks a day.

A programming language shouldn't change its accent in every major point release. This is also true about libraries, frameworks, and APIs.

Adhering to a process is like driving in the freeway, when everyone is going at sustainable developer velocity, the process is flowing. But it only takes one developer to stall in the middle of the process to back up and side rail the whole team, as one stalled car in the freeway backs up everybody for miles.

In the end, all software ends up alike, just a bunch of boxes wired together.

Software development teams are like Tolstoy's Family, there is one way to produce happy software, but unlimited number of ways to be down trodden frustrated and dysfunctional software.

Measuring programming progress by lines of code is like measuring a horror flick by the number of victims, for either case it's usually a gorefest.

As software developers, half the stuff we do is hacks for the other half.

Don't let the spec define and technology limit the user experience.

Software engineering wants to be a creative, like writing, and disciplined, like engineering, but it is often neither.

Remember that process is there to help people, not people to push along the process.

I've heard of greenfield app, now brownfield app. A brownfield is a project that is contaminated by poor practices but has the potential to be revived. What is next? Cloverfield?

The only thing I hate most other than writing unit tests, is refactoring, updating, and maintaining code that do not have unit tests.

In most applications you'll ever develop, optimization and scalability are just 10% of the job. Focus on the 90%, which is getting the thing to work correctly.

Learning a new platform requires you learn a whole set of known issues and limitations, and to discover new ones in the process.

Anyone eager to deploy a web application on a new framework is a masochist.

Backward compatibility is not just about code but also applies to usability. Backward compatible applies to user expectations.

Rails is not just opinionated software, it is fascist software.

The Java version of a Hello World example Web Services using Axis 2 is just about ten lines of code, 55 jars, and 20MB war file. Fail.

Everyone has ideas, ideas are a cheap commodity. Visualizing an idea for five minutes is not the same thing as working years to actualize it.

Not every cloud computing initiative has a silver lining.

Big O notation is used to describe computational complexity, I use Big H notation to describe the level of a hack. $H(n!)$ describes a MacGyver level hack. Similarly there is a Big F notation for level of FAIL.

In programming there is no force of nature or grace of God, just hacks of probability and redundancy.

Saying that it will be fixed in a future release does not actually fix the problem.



When hiring, I'll take someone that wants to do it over someone that has done it once and is willing to do it again until something better comes along. That is the difference between passion and bandwagon, career man and salary man, patriot and mercenary, love and prostitution.

If all things are equal and fully qualify, hire those that want to be there, contract those that just need to be somewhere.

It doesn't matter how smart you are, or how smart I think I am. Regression testing will prove us wrong.

I have heard that some teams follow the Software Development Life Cycle. But I have seen teams follow the Software Development Death March.

Recruit. Retain. Retrain.

Some entrepreneurs' idea of a business model is how much they can charge their customers not what they can do for them.

Some entrepreneurs are looking for a sexy business model, I'll stick to my hard working homely business model.

Finding a service provider: Good, Fast, Cheap. Pick two. Finding an employee: Passionate, Knowledgeable, Productive. Pick one.

When people are too busy to learn, they are not being productive.

Which is best, an extraordinary idea executed ordinary and ordinary idea executed extraordinary?

Learn to manage your manager.

You could only be excellent, you can't be half excellent

You can only stand in front of a client only if you know what you say you know.

You can't charge for testing, but it saves you time and money, it is an investment in quality and sanity.

The problem with frameworks is that you most often have a very narrow frame to work with.

Management has a way of over emphasizing the blatantly obvious.