2. Saying No

“Do; or do not. There is no trying.”

—Yoda

In the early ’70s, I, and two of my nineteen-year-old friends were working on a real-time accounting system for the Teamster’s union in Chicago for a company named ASC. If names like Jimmy Hoffa come to mind, they should. You didn’t mess around with the teamsters in 1971.

Our system was supposed to go live by a certain date. A *lot* of money was riding on that date. Our team had been working 60-, 70-, and 80-hour weeks to try to hold to that schedule.

A week before the go-live date we finally got the system put together in its entirety. There were lots of bugs and issues to deal with, and we frantically worked through the list. There was barely time to eat and sleep, let alone think.

Frank, the manager of ASC, was a retired Air Force colonel. He was one of those loud, in-your-face kind of managers. It was his way or the highway, and he’d put you on that highway by dropping you from 10,000 feet without a parachute. We nineteen year olds were barely able to make eye contact with him.

Frank said it had to be done by the date. That was all there was to it. The date would come, and we would be done. Period. No discussion. Over and out.

My boss, Bill, was a likeable guy. He'd been working with Frank for quite a few years and understood what was possible with Frank, and what was not. He told us that we were going live on the date, no matter what.

So we went live on the date. And it was a blazing disaster.

There were a dozen 300-baud, half-duplex terminals that connected Teamster's headquarters in Chicago to our machine thirty miles north in the suburbs. Each of those terminals locked up every 30 minutes or so. We had seen this problem before but had not simulated the traffic that the union data-entry clerks were suddenly slamming into our system.

To make matters worse, the tear sheets being printed on the ASR35 teletypes that were also connected to our system by 110-baud phone lines would freeze up in the middle of printing.

The solution to these freeze-ups was to reboot. So they'd have to get everybody whose terminal was still live to finish their work and then stop. When everyone was stopped, then they'd call us to reboot. The people who had been frozen would have to start over. And this was happening more than once per hour.

After half a day of this, the Teamster's office manager told us to shut the system down and not bring it up again until we had it working. Meanwhile, they had lost a half day of work and were going to have to re-enter it all using the old system.

We heard Frank's wails and roars all through the building. They went on for a long, long time. Then Bill, and our system's analyst Jalil, came to us and asked when we could have the system stable. I said, "four weeks."

The look on their faces was horror and then determination. "No," they said, "it must be running by Friday."

So I said, "Look, we just barely got this system to sort-of work last week. We need to shake down the troubles and issues. We need four weeks."

But Bill and Jalil were adamant. "No, it's really got to be Friday. Can you at least try?"

Then our team leader said, "OK, we'll try."

Friday was a good choice, The weekend load was a lot lower. We were able to find more problems and correct them before Monday came. Even

so, the whole house of cards nearly came tumbling down again. The freeze-up problems kept on happening once or twice a day. There were other problems too. But gradually, after a few more weeks, we got the system to the point where the complaints died down, and normal life looked like it might actually be possible.

And then, as I told you in the introduction, we all quit. And they were left with a real crisis on their hands. They had to hire a new batch of programmers to try to deal with the huge stream of issues coming from the customer.

Who can we blame this debacle on? Clearly, Frank's style is part of the problem. His intimidations made it difficult for him to hear the truth. Certainly Bill and Jalil should have pushed back on Frank much harder than they did. Certainly our team lead should not have caved in to the Friday demand. And certainly I should have continued to say "no" instead of getting in line behind our team lead.

Professionals speak truth to power. Professionals have the courage to say no to their managers.

How do you say no to your boss? After all, it's your *boss!* Aren't you supposed to do what your boss says?

No. Not if you are a professional.

Slaves are not allowed to say no. Laborers may be hesitant to say no. But professionals are *expected* to say no. Indeed, good managers crave someone who has the guts to say no. It's the only way you can really get anything done.

Adversarial Roles

One of the reviewers of this book truly hated this chapter. He said that it almost made him put the book down. He had built teams where there were no adversarial relationships; the teams worked together in harmony and without confrontation.

I'm happy for this reviewer, but I wonder if his teams are really as confrontation free as he supposes. And if they are, I wonder if they are as efficient as they could be. My own experience has been that the hard decisions are best made through the confrontation of adversarial roles.

Managers are people with a job to do, and most managers know how to do that job pretty well. Part of that job is to pursue and defend their objectives as aggressively as they can.

By the same token, programmers are also people with a job to do, and most of them know how to get that job done pretty well. If they are professionals they will pursue and defend *their* objectives as aggressively as *they* can.

When your manager tells you that the login page has to be ready by tomorrow, he is pursuing and defending one of his objectives. He's doing his job. If you know full well that getting the login page done by tomorrow is impossible, then you are not doing your job if you say "OK, I'll try." The only way to do your job, at that point, is to say "No, that's impossible."

But don't you have to do what your manager says? No, your manager is counting on you to defend your objectives as aggressively as he defends his. That's how the two of you are going to get to *the best possible outcome*.

The best possible outcome is the goal that you and your manager share. The trick is to find that goal, and that usually takes negotiation.

Negotiation can sometimes be pleasant.

> Mike: "Paula, I need the login page done by tomorrow."
>
> Paula: "Oh, wow! That soon? Well, OK, I'll try."
>
> Mike: "OK, that's great. Thanks!"

That was a nice little conversation. All confrontation was avoided. Both parties left smiling. Nice.

But both parties were behaving unprofessionally. Paula knows full well that the login page is going to take her longer than a day, so she's just lying. She might not think of it as a lie. Perhaps she thinks she *actually will try*, and maybe she holds out some meager hope that she'll actually get it done. But in the end, it's still a lie.

Mike, on the other hand, accepted the "I'll try" as "Yes." That's just a dumb thing to do. He should have known that Paula was trying to avoid confrontation, so he should have pressed the issue by saying, "You seem hesitant. Are you sure you can get it done tomorrow?"

Here's another pleasant conversation.

Mike: "Paula, I need the login page done by tomorrow."

Paula: "Oh, sorry Mike, but it's going to take me more time than that."

Mike: "When do you think you can have it done?"

Paula: "How about two weeks from now?"

Mike: (scribbles something in his daytimer) "OK, thanks."

As pleasant as that was, it was also terribly dysfunctional and utterly unprofessional. Both parties failed in their search for the best possible outcome. Instead of asking whether two weeks would be OK, Paula should have been more assertive: "It's going to take me two weeks, Mike."

Mike, on the other hand, just accepted the date without question, as though his own objectives didn't matter. One wonders if he's not going to simply report back to his boss that the customer demo will have to be postponed because of Paula. That kind of passive-aggressive behavior is morally reprehensible.

In all these cases neither party has pursued a common acceptable goal. Neither party has been looking for the best possible outcome. Let's try this.

Mike: "Paula, I need the login page done by tomorrow."

Paula: "No, Mike, that's a two-week job."

Mike: "Two weeks? The architects estimated it at three days and it's already been five!"

Paula: "The architects were wrong, Mike. They did their estimates before product marketing got hold of the requirements. I've got at least ten more days of work to do on this. Didn't you see my updated estimate on the wiki?"

Mike: (looking stern and trembling with frustration) "This isn't acceptable Paula. Customers are coming for a demo tomorrow, and I've got to show them the login page working."

Paula: "What part of the login page do you need working by tomorrow?"

Mike: "I need *the login page!* I need to be able to *log in*."

Paula: “Mike, I can give you a mock-up of the login page that will let you log in. I’ve got that working now. It won’t actually check your username and password, and it won’t email a forgotten password to you. It won’t have the company news banner “Times-squaring” around the top of it, and the help button and hover text won’t work. It won’t store a cookie to remember you for next time, and it won’t put any permission restrictions on you. But you’ll be able to log in. Will that do?”

Mike: “I’ll be able to log in?”

Paula: “Yes, you’ll be able to log in.”

Mike: “That’s great Paula, you’re a life saver!” (walks away pumping the air and saying “Yes!”)

They reached the best possible outcome. They did this by saying no and then working out a solution that was mutually agreeable to both. They were acting like professionals. The conversation was a bit adversarial, and there were a few uncomfortable moments, but that’s to be expected when two people assertively pursue goals that aren’t in perfect alignment.

What about the Why?

Perhaps you think that Paula should have explained *why* the login page was going to take so much longer. My experience is that the *why* is a lot less important than the *fact*. That fact is that the login page will require two weeks. Why it will take two weeks is just a detail.

Still, knowing why might help Mike to understand, and therefore to accept, the fact. Fair enough. And in situations where Mike has the technical expertise and temperament to understand, such explanations might be useful. On the other hand, Mike might disagree with the conclusion. Mike might decide that Paula was doing it all wrong. He might tell her that she doesn’t need all that testing, or all that reviewing, or that step 12 could be omitted. Providing too much detail can be an invitation for micro-management.

High Stakes

The most important time to say no is when the stakes are highest. The higher the stakes, the more valuable no becomes.

This should be self-evident. When the cost of failure is so high that the survival of your company depends upon it, you must be absolutely determined to give your managers the best information you can. And that often means saying no.

> Don (Director of Development): “So, our current estimate for completion of the Golden Goose project is twelve weeks from today, with an uncertainty of plus or minus five weeks.”
>
> Charles (CEO): (sits glaring for fifteen seconds as his face reddens) “Do you mean to sit there and tell me that we might be seventeen weeks from delivery?”
>
> Don: “That’s possible, yes.”
>
> Charles: (stands up, Don stands up a second later) “Damm it Don! This was supposed to be done three weeks ago! I’ve got Galitron calling me every day wondering where their frakking system is. I am *not* going to tell them that they have to wait another four months? You’ve got to do better.”
>
> Don: Chuck, I told you *three months ago*, after all the layoffs, that we’d need four more months. I mean, Christ Chuck, you cut my staff twenty percent! Did you tell Galitron then that we’d be late?”
>
> Charles: “You know damned well I didn’t. We can’t afford to lose that order Don. (Charles pauses, his face goes white) Without Galitron, we’re really hosed. You know that, don’t you? And now with this delay, I’m afraid . . . What will I tell the board? (He slowly sits back down in his seat, trying not to crumble.) Don, you’ve got to do better.”
>
> Don: “There’s nothing I can do Chuck. We’ve been through this already. Galitron won’t cut scope, and they won’t accept any interim releases. They want to do the installation once and be done with it. I simply cannot do that any faster. It’s *not* going to happen.”
>
> Charles: “Damn. I don’t suppose it would matter if I told you your job was at stake.”
>
> Don: “Firing me isn’t going to change the estimate, Charles.”

Charles: “We’re done here. Go back to your team and keep this project moving. I’ve got some very tough phone calls to make.”

Of course, Charles should have told Galitron no three months ago when he first found out about the new estimate. At least now he’s doing the right thing by calling them (and the board). But if Don hadn’t stuck to his guns, those calls might have been delayed even longer.

Being a “Team Player”

We’ve all heard how important it is to be a “team player.” Being a team player means playing your position as well as you possibly can, and helping out your teammates when they get into a jam. A team-player communicates frequently, keeps an eye out for his or her teammates, and executes his or her own responsibilities as well as possible.

A team player is not someone who says yes all the time. Consider this scenario:

Paula: “Mike, I’ve got those estimates for you. The team agrees that we’ll be ready to give a demo in about eight weeks, give or take one week.”

Mike: “Paula, we’ve already scheduled the demo for six weeks from now.”

Paula: “Without hearing from us first? Come on Mike, you can’t push that on us.”

Mike: “It’s already done.”

Paula: (sigh) “OK, look, I’ll go back to the team and find out what we can safely deliver in six weeks, but it won’t be the whole system. There’ll be some features missing, and the data load will be incomplete.”

Mike: “Paula, the customer is expecting to see a complete demo.”

Paula: “That’s not going to happen Mike.”

Mike: “Damn. OK, work up the best plan you can and let me know tomorrow.”

Paula: “That I can do.”

Mike: “Isn’t there something you can do to bring this date in? Maybe there’s a way to work smarter and get creative.”

Paula: “We’re all pretty creative, Mike. We’ve got a good handle on the problem, and the date is going to be eight or nine weeks, not six.”

Mike: “You could work overtime.”

Paula: “That just makes us go slower, Mike. Remember the mess we made last time we mandated overtime?”

Mike: “Yeah, but that doesn’t have to happen this time.”

Paula: “It’ll be just like last time, Mike. Trust me. It’s going to be eight or nine weeks, not six.”

Mike: “OK, get me your best plan, but keep thinking about how to get it done in six weeks. I know you guys’ll figure out something.”

Paula: “No, Mike, we won’t. I’ll get you a plan for six weeks, but it will be missing a lot of features and data. That’s just how it’s going to be.”

Mike: “OK, Paula, but I bet you guys can work miracles if you try.”

(Paula walks away shaking her head.)

Later, in the Director’s strategy meeting ...

Don: “OK Mike, as you know the customer is coming in for a demo in six weeks. They’re expecting to see everything working.”

Mike: “Yes, and we’ll be ready. My team is busting their butts on this and we’re going to get it done. We’ll have to work some overtime, and get pretty creative, but we’ll make it happen!”

Don: “It’s great that you and your staff are such team players.”

Who were the *real* team players in this scenario? Paula was playing for the team, because she represented what could, and could not, be done to the best of her ability. She aggressively defended her position, despite the wheedling and cajoling from Mike. Mike was playing on a team of one. Mike is for Mike. He’s clearly not on Paula’s team because he just committed her to something she explicitly said she couldn’t do. He’s not

on Don's team either (though he'd disagree) because he just lied through his teeth.

So why did Mike do this? He wanted Don to see him as a team player, and he has faith in his ability to wheedle and manipulate Paula into *trying* for the six-week deadline. Mike is not evil; he's just too confident in his ability to get people to do what he wants.

Trying

The worst thing Paula could do in response to Mike's manipulations is say "OK, we'll try." I hate to channel Yoda here, but in this instance he is correct. *There is no trying.*

Perhaps you don't like that idea? Perhaps you think *trying* is a positive thing to do. After all, would Columbus have discovered America if he hadn't tried?

The word *try* has many definitions. The definition I take issue with here is "to apply extra effort." What extra effort could Paula apply to get the demo ready in time? If there *is* extra effort she could apply, then she and her team must not have been applying all their effort before. They must have been holding some effort in reserve.[1]

The promise to try is an admission that you've been holding back, that you have a reservoir of extra effort that you can apply. The promise to try is an admission that the goal is attainable through the application of this extra effort; moreover, it is a commitment to apply that extra effort to achieve the goal. Therefore, by promising to try you are committing to succeed. This puts the burden on you. If your "trying" does not lead to the desired outcome, you will have failed.

Do you have an extra reservoir of energy that you've been holding back? If you apply these reserves, will you be able to meet the goal? Or, by promising to try are you simply setting yourself up for failure?

By promising to try you are promising to change your plans. After all, the plans you had were insufficient. By promising to try you are saying that you have a new plan. What is that new plan? What change will you make to your behavior? What different things are you going to do because now you are "trying"?

If you don't have a new plan, if you don't make a change to your behavior, if you do everything exactly as you would have before you promised to "try," then what does trying mean?

If you are not holding back some energy in reserve, if you don't have a new plan, if you aren't going to change your behavior, and if you are reasonably confident in your original estimate, then promising to try is fundamentally dishonest. You are *lying*. And you are probably doing it to save face and to avoid a confrontation.

Paula's approach was much better. She continued to remind Mike that the team's estimate was uncertain. She always said "eight or nine weeks." She stressed the uncertainty and never backed off. She never suggested that there might be some extra effort, or some new plan, or some change in behavior that could reduce that uncertainty.

Three weeks later ...

> Mike: "Paula, the demo is in three weeks, and the customers are demanding to see FILE UPLOAD working."
>
> Paula: "Mike, that's not on the list of features we agreed to."
>
> Mike: "I know, but they're demanding it."
>
> Paula: "OK, that means that either SINGLE SIGN-ON or BACKUP will have to be dropped from the demo."
>
> Mike: "Absolutely not! They're expecting to see those features working as well!"
>
> Paula: "So then, they are expecting to see every feature working. Is that what you are telling me? I told you that wasn't going to happen."
>
> Mike: "I'm sorry Paula, but the customer just won't budge on this. They want to see it all."
>
> Paula: "That's not going to happen, Mike. It's just not."
>
> Mike: "Come on Paula, can't you guys at least *try*?"
>
> Paula: "Mike, I could *try* to levitate. I could *try* to change lead in to gold. I could *try* to swim across the Atlantic. Do you think I'd succeed?"
>
> Mike: "Now you're being unreasonable. I'm not asking for the *impossible*."

Paula: “Yes, Mike, you *are*.”

(Mike smirks, nods, and turns to walk away.)

Mike: “I’ve got faith in you Paula; I know you won’t let me down.”

Paula: (speaking to Mike’s back) “Mike, you’re dreaming. This *is not* going to end well.”

(Mike just waves without turning around.)

Passive Aggression

Paula’s got an interesting decision to make. She suspects that Mike is not telling Don about her estimates. She could just let Mike walk off the end of the cliff. She could make sure that copies of all the appropriate memos were on file, so that when the disaster strikes she can show *what* she told Mike, and *when* she told him. This is passive aggression. She’d just let Mike hang himself.

Or, she could try to head off the disaster by communicating directly with Don. This is risky, to be sure, but it’s also what being a team player is really all about. When a freight train is bearing down on you and you are the only one who can see it, you can either step quietly off the track and watch everyone else get run over, or you can yell “Train! Get off the track!”

Two days later ...

Paula: “Mike, have you told Don about my estimates? Has he told the customer that the demo will not have the FILE UPLOAD feature working?”

Mike: “Paula, you said you’d get that working for me.”

Paula: “No, Mike, I didn’t. I told you that it was impossible. Here’s a copy of the memo I sent you after our talk.”

Mike: “Yeah, but you were going to *try* anyway, right?”

Paula: “We’ve already had that discussion Mike. Remember, gold and lead?”

Mike: (sighs) “Look, Paula, you’ve just got to do it. You just have to. Please do whatever it takes, but you just have to make this happen for me.”

Paula: “Mike. You’re wrong. I don’t have to make this happen for you. What I *have* to do, if you don’t, is tell Don.”

Mike: “That’d be going over my head, you wouldn’t do that.”

Paula: “I don’t want to Mike, but I will if you force me.”

Mike: “Oh, Paula . . .”

Paula: “Look, Mike, the features *aren’t* going to get done in time for the demo. You need to get this into your head. Stop trying to convince me to work harder. Stop deluding yourself that I’m somehow going to pull a rabbit out of a hat. Face the fact that you have to tell Don, and you have to tell him *today*.”

Mike: (Eyes wide) “Today?”

Paula: “Yes, Mike. Today. Because tomorrow I expect to have a meeting with you and Don about which features to include in the demo. If that meeting doesn’t happen tomorrow, then I will be forced to go to Don myself. Here’s a copy of the memo that explains just that.”

Mike: “You’re just covering your ass!”

Paula: “Mike, I’m trying to cover *both* our asses. Can you imagine the debacle if the customer comes here expecting a full demo and we can’t deliver?”

What happens in the end to Paula and Mike? I’ll leave it to you to work out the possibilities. The point is that Paula has behaved very professionally. She has said no at all the right times, and in all the right ways. She said no when pushed to amend her estimates. She said no when manipulated, cajoled, and begged. And, most importantly, she said no to Mike’s self-delusion and inaction. Paula was playing for the team. Mike needed help, and she used every means in her power to help him.

The Cost of Saying Yes

Most of the time we want to say yes. Indeed, healthy teams strive to find a way to say yes. Manager and developers in well-run teams will negotiate with each other until they come to a mutually agreed upon plan of action.

But, as we've seen, sometimes the only way to get to the *right* yes is to be unafraid so say no.

Consider the following story that John Blanco posted on his blog.[2] It is reprinted here with permission. As you read it, ask yourself when and how he should have said no.

Is Good Code Impossible?

When you hit your teenage years you decide you want to be a software developer. During your high school years, you learn how to write software using object-oriented principles. When you graduate to college, you apply all the principles you've learned to areas such as artificial intelligence or 3D graphics.

And when you hit the professional circuit, you begin your never-ending quest to write commercial-quality, maintainable, and "perfect" code that will stand the test of time.

Commercial quality. Huh. That's pretty funny.

I consider myself lucky, I *love* design patterns. I like studying the theory of coding perfection. I have no problem starting up an hour-long discussion about why my XP partner's choice of inheritance hierarchy is wrong—that HAS-A is better than IS-A in so many cases. But something has been bugging me lately and I am wondering something . . .

. . . Is good code impossible in modern software development?

The Typical Project Proposal

As a full-time contract developer (and part-time), I spend my days (and nights) developing mobile applications for clients. And what I've learned over the many years I've been doing this is that the demands of client work preclude me from writing the real quality apps that I'd like.

Before I begin, let me just say it's not for a lack of trying. I love the topic of clean code. I don't know anyone who pursues that perfect software design like I do. It's the execution that I find more elusive, and not for the reason you think.

Here, let me tell you a story.

Towards the end of last year, a fairly well-known company put out an RFP (Request for Proposal) to have an app built for them. They're a huge retailer, but for the sake of anonymity let's call them Gorilla Mart. They say they need to create an iPhone presence and would like an app produced for them by Black Friday. The catch? It's already November 1st. That leaves just under 4 weeks to create the app. Oh, and at this time Apple is still taking two weeks to approve apps. (Ah, the good old days.) So, wait, this app has to be written in . . . TWO WEEKS?!?!

Yes. We have two weeks to write this app. And, unfortunately, we've won the bid. (In business, client importance matters.) This is going to happen.

> "But it's OK," Gorilla Mart Executive #1 says. "The app is simple. It just needs to show users a few products from our catalog and let them search for store locations. We already do it on our site. We'll give you the graphics, too. You can probably—what's the word—yeah, hardcode it!"
>
> Gorilla Mart Executive #2 chimes in. "And we just need a couple of coupons the user can show at the cash register. The app will be a throwaway. Let's get it out the door, and then for Phase II we'll do something bigger and better from scratch."

And then it's happening. Despite years of constant reminders that every feature a client asks for will always be more complex to write than it is to explain, you go for it. You really believe that this time it really can be done in two weeks. Yes! We can do this! This time it's different! It's just a few graphics and a service call to get a store location. XML! No sweat. We can do this. I'm pumped! Let's go!

It takes just a day for you and reality to once again make acquaintance.

Me: So, can you give me the info I need to call your store location web service?

The Client: What's a web service?

Me: .....

And that's exactly how it happened. Their store location service, found right where it's supposed to be on the top-right corner of their web site, is not a web service. It's generated by Java code. Ix-nay with the API-ay. And to boot, it's hosted by a Gorilla Mart strategic partner.

Enter the nefarious "3rd party."

In client terms, a "3rd party" is akin to Angelina Jolie. Despite the promise that you'll be able to have an enlightening conversation over a nice meal and hopefully hook up afterwards ... sorry, it ain't happenin'. You're just gonna have to fantasize about it while you take care of business yourself.

In my case, the only thing I was able to wrestle out of Gorilla Mart was a current snapshot of their current store listings in an Excel file. I had to write the store location search code from scratch.

The double-whammy came later that day: They wanted the product and coupon data online so it could be changed weekly. There goes hardcoding! Two weeks to write an iPhone app have now become two weeks to write an iPhone app, a PHP backend, and integrate them togeth—. . . What? They want me to handle QA, too?

To make up for the extra work, the coding will have to go a little faster. Forget that abstract factory. Use a big fat for loop instead of the composite, there's no time!

Good code has become impossible.

Two Weeks to Completion

Let me tell you, that two weeks was pretty miserable. First, two of the days were eliminated due to all-day meetings for my next project. (That amplifies how short a time frame this was going to be.) Ultimately, I really had eight days to get things

done. The first week I worked 74 hours and the next week . . . God . . . I don't even recall, it's been eradicated from my synapses. Probably a good thing.

I spent those eight days writing code in a fury. I used all the tools available to me to get it done: copy and paste (AKA reusable code), magic numbers (avoiding the duplication of defining constants and then, gasp!, retyping them), and absolutely NO unit tests! (Who needs red bars at a time like this, it'd just demotivate me!)

It was pretty bad code and I never had time to refactor. Considering the time frame, however, it was actually pretty stellar, and it was "throwaway" code after all, right? Does any of this sound familiar? Well just wait, it gets better.

As I was putting the final touches on the app (the final touches being writing the entirety of the server code), I started to look at the codebase and wondered if maybe it was worth it. The app was done after all. I survived!

> "Hey, we just hired Bob, and he's very busy and he couldn't make the call, but he says we should be requiring users to provide their email addresses to get the coupons. He hasn't seen the app, but he thinks this would be a great idea! We also want a reporting system to get those emails from the server. One that's nice and not too expensive. (Wait, that last part was Monty Python.) Speaking of coupons, they need to be able to expire after a number of days we specify. Oh, and ..."

Let's step back. What do we know about what good code is? Good code should be extendable. Maintainable. It should lend itself to modification. It should read like prose. Well, this wasn't good code.

Another thing. If you want to be a better developer, you must always keep this inevitably in mind: The client will always extend the deadline. They will always want more features. They

will always want change—LATE. And here's the formula for what to expect:

$$(\#\text{ of Executives})^2$$
$$+ 2 * \#\text{ of New Executives}$$
$$+ \#\text{ of Bob's Kids}$$
$$= \text{DAYS ADDED AT LAST MINUTE}$$

Now, executives are decent people. I think. They provide for their family (assuming Satan has approved of their having one). They want the app to succeed (promotion time!). The problem is that they all want a direct claim to the project's success. When all is said and done, they all want to point at some feature or design decision they can each call their very own.

So, back to the story, we added a couple more days to the project and got the email feature done. And then I collapsed from exhaustion.

The Clients Never Care as Much as You Do

The clients, despite their protestations, despite their apparent urgency, never care as much as you do about the app being on time. The afternoon that I dubbed the app completed, I sent an email with the final build to all the stakeholders, Executives (hiss!), managers, and so on. "IT IS DONE! I BRING YOU V1.0! PRAISE THY NAME." I hit Send, lay back in my chair, and with a smug grin began to fantasize how the company would run me up onto their shoulders and lead a procession down 42nd Street while I was crowned "Greatest Developer Ev-ar." At the very least, my face would be on all their advertising, right?

Funny, they didn't seem to agree. In fact, I wasn't sure what they thought. I heard nothing. Not a peep. Turns out, the folks at Gorilla Mart were eager to and had already moved on to the next thing.

You think I lie? Check this out. I pushed to the Apple store without filling in an app description. I had requested one from Gorilla Mart, and they hadn't gotten back to me and there was no time to wait. (See previous paragraph.) I wrote them again. And

again. I got some of our own management on it. Twice I heard back and twice I was told, “What did you need again?” I NEED THE APP DESCRIPTION!

One week later, Apple started testing the app. This is usually a time of joyousness, but it was instead a time for mortal dread. As expected, later in the day the app was rejected. It was about the saddest, poorest excuse to allow a rejection I can imagine: “App is missing an app description.” Functionally perfect; no app description. And for this reason Gorilla Mart didn’t have their app ready for Black Friday. I was pretty upset.

I’d sacrificed my family for a two-week super sprint, and no one at Gorilla Mart could be bothered to create an app description given a week of time. They gave it to us an hour after the rejection—apparently that was the signal to get down to business.

If I was upset before, I would become livid a week and a half after that. You see, they still hadn’t gotten us real data. The products and coupons on the server were fake. Imaginary. The coupon code was 1234567890. You know, phoney baloney. (Bologna is spelled baloney when used in that context, BTW.)

And it was that fateful morning that I checked the Portal and THE APP WAS AVAILABLE! Fake data and all! I cried out in abject horror and called up whoever I could and screamed, “I NEED THE DATA!” and the woman on the other end asked me if I needed fire or police, so I hung up on 911. But then I called Gorilla Mart and was like, “I NEED DATA!” And I’ll never forget the response:

> Oh, hey there John. We have a new VP and we’ve decided not to release. Pull it off the App Store, would you?

In the end, it turned out that at least 11 people registered their email addresses in the database, which meant there were 11 people that could potentially walk into a Gorilla Mart with a fake iPhone coupon in tow. Boy, that might get ugly.

When it was all said and done, the client had said one thing correctly all along: The code was a throwaway. The only problem is, it was never released in the first place.

Result? Rush to Complete, Slow to Market

The lesson in the story is that your stakeholders, whether an external client or internal management, have figured out how to get developers to write code quickly. Effectively? No. Quickly? Yes. Here's how it works:

> • **Tell the developer the app is simple.** This serves to pressure the development team into a false frame of mind. It also gets the developers to start working earlier, whereby they ...
>
> • **Add features by faulting the team for not recognizing their necessity.** In this case, the hardcoded content was going to require app updates to change. How could I not realize that? I did, but I'd been handed a false promise earlier, that's why. Or a client will hire "a new guy" who's recognized there is some obvious omission. One day a client will say they just hired Steve Jobs and can we add alchemy to the app? Then they'll ...
>
> • **Push the deadline. Over and over.** Developers work their fastest and hardest (and BTW are at their most error prone, but who cares about that, right?) with a couple days to go on a deadline. Why tell them you can push the date out further while they're being so productive? Take advantage of it! And so it goes, a few days are added, a week is added, just when you had worked a 20-hour shift to get everything just right. It's like a donkey and carrot, except you're not treated as well as the donkey.

It's a brilliant playbook. Can you blame them for thinking it works? But they don't see the God-awful code. And so it happens, time and again, despite the results.

> In a globalized economy, where corporations are held to the almighty dollar and raising the stock price involves layoffs, overworked staffs, and offshoring, this strategy I've shown you of cutting developer costs is making good code obsolete. As developers, we're going to be asked/told/conned into writing twice the code in half the time if we're not careful.

Code Impossible

In the story when John asks "Is good code impossible?", he is really asking "Is professionalism impossible?" After all, it wasn't just the code that suffered in his tale of dysfunction. It was his family, his employer, his customer, and the users. *Everybody* lost[3] in this adventure. And they lost due to unprofessionalism.

So who was acting unprofessionally? John makes it clear that he thinks it was the executives at Gorilla Mart. After all, his playbook was a pretty clear indictment of their bad behavior. But *was* their behavior bad? I don't think so.

The folks at Gorilla Mart wanted the option to have an iPhone app on Black Friday. They were willing to pay to have that option. They found someone willing to provide that option. So how can you fault them?

Yes, it's true, there were some communications failures. Apparently the executives didn't know what a web service really was, and there were all the normal issues of one part of a big corporation not knowing what another part is doing. But all that should have been expected. John even admits as much when he says: "Despite years of constant reminders that every feature a client asks for will always be more complex to write than it is to explain. . ."

So if the culprit was not Gorilla Mart, then who?

Perhaps it was John's direct employer. John didn't say this explicitly, but there was a hint when he said, parenthetically, "In business, client importance matters." So did John's employer make unreasonable promises to Gorilla Mart? Did they put pressure on John, directly or indirectly, to

make those promises come true? John doesn't say this, so we can only wonder.

Even so, where is John's responsibility in all of this? I put the fault squarely on John. John is the one who accepted the initial two-week deadline, knowing full well that projects are usually more complex than they sound. John is the one who accepted the need to write the PHP server. John is the one who accepted the email registration, and the coupon expiration. John is the one who worked 20-hour days and 90-hour weeks. John is the one who subtracted himself from his family and his life to make this deadline.

And why did John do this? He tells us in no uncertain terms: "I hit Send, lay back in my chair, and with a smug grin began to fantasize how the company would run me up onto their shoulders and lead a procession down 42nd Street while I was crowned "Greatest Developer Ev-ar." In short, John was trying to be a hero. He saw his chance for accolades, and he went for it. He leaned over and grabbed for the brass ring.

Professionals are often heroes, but not because they try to be. Professionals become heroes when they get a job done well, on time, and on budget. By trying to become the man of the hour, the savior of the day, John was not acting like a professional.

John should have said no to the original two-week deadline. Or if not, then he should have said no when he found there was no web service. He should have said no to the request for email registration and coupon expiration. He should have said no to anything that would require horrific overtime and sacrifice.

But most of all, John should have said no to his own internal decision that the only way to get this job done on time was to make a big mess. Notice what John said about good code and unit tests:

"To make up for the extra work, the coding will have to go a little faster. Forget that abstract factory. Use a big fat for loop instead of the composite, there's no time!"

And again:

"I spent those eight days writing code in a fury. I used all the tools available to me to get it done: copy-and-paste (AKA reusable code), magic numbers (avoiding the duplication of defining constants and then, gasp!,

retyping them), and absolutely NO unit tests! (Who needs red bars at a time like this, it'd just demotivate me!)"

Saying yes to those decisions was the real crux of the failure. John accepted that the only way to succeed was to behave unprofessionally, so he reaped the appropriate reward.

That may sound harsh. It's not intended that way. In previous chapters I described how I've made the same mistake in my career, more than once. The temptation to be a hero and "solve the problem" is huge. What we all have to realize is that saying yes to dropping our professional disciplines is *not* the way to solve problems. Dropping those disciplines is the way you create problems.

With that, I can finally answer John's initial question:

"Is good code impossible? Is professionalism impossible?"

Answer: I say *no*.