

# Management Means Never Having to Say You're Sorry

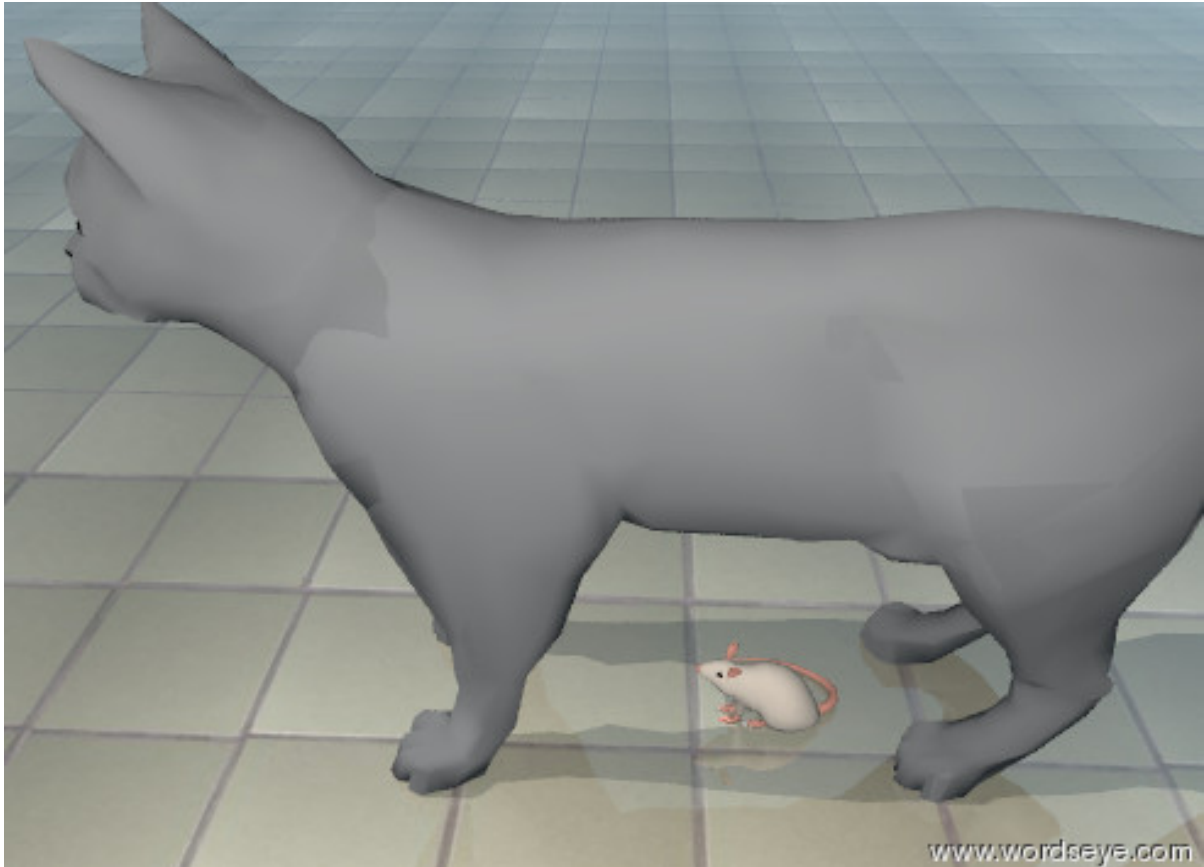
by Philip Chu

## Table of contents

1 Publication Information.....	2
2 Know Your Stuff.....	2
3 Good Management is Risk Management.....	4
4 Stay On Target.....	7
5 You're the Boss.....	9
6 Lead By Example.....	10

## 1. Publication Information

Copyright ©2004-2007 by Philip Chu All rights reserved.



"I have people skills; I am good at dealing with people. Can't you understand that? What the hell is wrong with you people?" - Tom Smykowski, in *Office Space*

## 2. Know Your Stuff

The saying goes, those who can't do, teach. The same could be said of management. (And those who can't manage, consult? But let's leave that for another essay...) In any case, I've often complained long and bitterly about software project management, and a coworker once responded, "Phil, you're always right". I don't know if he was serious, but in the spirit of that encouraging or sarcastic endorsement, here are my recommended practices for managing software projects.

In many cases, "project manager" is a Microsoft Project manager whose apparent sole responsibility is to maintain a Gantt chart. (Sometimes even that is too advanced - I've sat in meetings watching people fumble with an Excel spreadsheet) But there's got to be more to project management, or any idiot could do it.

## **2.1. Know the Technology**

The best way to gain the respect of your engineering staff is to understand the technology they're using. This isn't the same as thinking you understand. A little knowledge is a dangerous thing, especially the latest knowledge gleaned from buzzword-laden articles in business magazines.

- At an early stage on one large application running in Lisp, I mentioned to my boss that I thought garbage collection was causing some performance problems. From then on, every time he saw a slowdown, he'd ponder, "hmm, looks like garbage collection."

As a manager, it's unlikely that you know more about everything than everyone on your team, even if you're the chief high tech mucky muck of your organization. So take the opportunity to learn from your people.

- I have a friend who graduated with a PhD from a well-regarded computer science department and yet was treated to an impromptu lecture on computer simulation from his company's president, who had lesser engineering credentials from over a decade ago. Not surprisingly, my friend was rather offended.

Worse is when your lack of expertise causes you to miss problems in the project design and execution.

- As a technical lead for a subsidiary of PRIA, a supplier of semiconductor fabrication equipment, I correctly chose XML as a platform-neutral format for our fab scheduling system, but didn't learn enough about XML until after the project was delivered to realize that we didn't do a very good job of sticking to XML (e.g. no DTD). Too bad - this was back when XML was pretty new stuff, so if I'd really known what I was doing, maybe I could have defined an industry standard format for semiconductor fab automation.

## **2.2. Master Your Domain**

Even if you don't know much about the engineering aspects of your project, you'll have value and get some respect if you know the application domain.

- As a development manager at Nichimen Graphics, a computer graphics software company, I visited one of our tool vendors and gave them an impromptu demo with the

intent of getting them excited about supporting our product. Unfortunately, I couldn't figure out what more to do after creating a cube and making it spin. Should have rehearsed, I guess.

Knowledge of your application domain is important in designing a useful product, supporting your customers, and avoiding embarrassment.

### **2.3. Know Management**

You should also be familiar with the state of the art in software project management.

- If there's one classic textbook on software engineering, it's Fred Brook's *The Mythical Man Month*, and if there's one classic software engineering principle anyone remembers from that book, it's the rule that adding more engineers to a late project just slows it down. I don't know if anyone I've ever worked for has read that book, but I have been on more than one project where panicked management piled on additional staff.

Of course, software engineering is just as trend-filled as other engineering fields (TQA, Six Sigma, ISO 9000...), and over-hyping is the rule, but you can't intelligently discuss practices and choose among them and apply them without reading up on them first. And if you don't know management stuff, what do you bring to the table - charisma? You're just an engineer who's not doing any engineering.

## **3. Good Management is Risk Management**

The most crucial aspect of project management is risk management. Every decision made in a project should be weighed against how much risk it adds to the delay or failure of the project.

### **3.1. If I Had a Nickel....**

I've often joked that every feature request should be accompanied by a dollar, but I betcha if managers had to whip out their own billfold and pull out a dollar for those "important" requests, the feature set would shrink dramatically.

- I was working on a submarine simulator that, up to that point, only had navigation controls. When our manager said he wanted to fire a torpedo for tomorrow's demo, we hastily improvised a reasonable facsimile. Encouraged by that, the manager innocently asked ten minutes before the demo whether he could fire a cruise missile. One of my coworkers grabbed him and pulled him out of the room before I could say anything.

There is no free lunch - everything has a cost, but it's usually felt by the poor slob who ends

up working the weekend rather than the person in management or marketing who belatedly said "we really need this".

### **3.2. Scotty Doesn't Work Here**

I blame Star Trek. Aliens are always humanoid and speak English, and the offworld females find Kirk irresistible - no one believes that's the stuff of reality. But the last fifteen minutes of every episode where Scotty or Geordi jury-rigs an engineering solution based on a never-tested hypothesis, and it always works just in time to avert disaster.

- I knew I was in a Star Trek moment on a military distributed interactive simulation project (basically a big multiplayer video game) when the project manager actually referred to me as "Geordi". Sitting in front of a mock submarine console, wearing submarine coveralls while the PR cameras rolled and Navy VIP's watched, I pushed the "start simulation" button that would connect us with the entire wide-area networked simulation and watched the system core dump as it got flooded with packets from the rest of the exercise. The project manager calmly informed everyone that we had "technical difficulties" and to "stand down", while I tried not to puke. I fixed the problem in the standard episode final fifteen minutes by deciding we didn't really care what anyone else was doing and turning off our system's data read from the outside world.

This project had the all the makings of a disaster - I cooled my heels for two months waiting for a security clearance, the networking hardware was two months late (it was a seven month project with a hard deadline), the laboratory had no experience on this type of project, and one of the three engineers working on the project didn't do anything (but amazingly kept repeating "I'll have it done tomorrow" for months). And then I spent the last month of the project working while flu-ridden. Our participation in the exercise was hailed as a success (evident when all the senior managers who had kept a healthy distance away suddenly show up), but I don't think I could physically take any more successes like that.

In real life, the Starship Enterprise would blow up every time - the episode-saving idea would be faulty, or Scotty (or Geordi or whoever, depending on which series you watch) is out sick, the warp core isn't performing up to spec, Jean-Luc didn't get the memo, etc. Depending on programmer heroics to pull you through the crunch time, and even worse, building such a crunch time into the schedule, is a recipe for disaster. It's fun to remember the successes, but the result is usually failure.

### **3.3. Get That Fresh and Clean Feeling**

Make sure you build the product regularly and from scratch. A common practice is to do this

as an automated nightly build. If you don't do it, then you never know if you can really do it.

- A group developing software for use with the Hubble Space Telescope decided to rebuild the currently-deployed version as a reference for our research in designing the next version. But the previous build had been made years ago, and there had been enough VMS Fortran compiler changes that the old software didn't compile anymore. If we actually had to fix a bug in the old software anytime in the past few years, we would have been out of luck.

It's no exaggeration to say that the fate of your company could depend on the ability to make a clean build

- One tool vendor I used to work with went Chapter 11 and at least one rescue acquisition failed because they couldn't demonstrate a successful clean build from the source code.

Having a regular automated build proves that you have a functioning build procedure set down in a script, not dependent on some sequence of magical incantations resident in one build engineer's head.

### **3.4. Let the Code Set**

I've never seen a code freeze really frozen to my satisfaction - there's always temptation to keep adding "just one more thing" until the final deadline. But instituting a freeze is better than coding pell-mell until the end, a sure recipe for a broken release.

- Game development is particularly notorious for crunch times, which poses greater risk as the projects get more complex. I was on one project where the last level was scheduled to be completed just before the game was to go "gold". Given my own stand was that asset creation had to be completed a month before the the release date, management did it's best to reassure me "it'll be all right", and indeed we did just complete it on time. I found out later that the level export performed on the day of the disc burn was missing key optimizations, so the game probably would be running at least twice as fast if we hadn't cut it so close.

The point is not to completely stop development, it's to slow it down in a deliberate manner to mitigate risk. It's really more a code "gel" than a code freeze. The alpha milestone should mark the feature freeze point - any feature that is not functional yet should not go in. Once the beta milestone is reached, every potential bug fix should be evaluated according to its benefits and risks.

Code isn't the only thing that should be frozen near the end of a project. The development environment should be stable, tool, i.e. you should have a tool freeze. New compilers,

libraries and runtime engines can introduce new bugs.

### **3.5. Know Your Demo**

Risk management is not just important for releases and milestone deliveries. Even demos should not be taken for granted.

- One company president kept bringing in bigwigs from the parent corporation for impromptu demos of an in-development product, even after I asked him to at least give me a couple of hours notice so I could make sure have the code running. Eventually, he strolled in, tried to run the product, crashed it and helplessly tried to restart it with the VIP watching and me studiously ignoring the proceedings. I almost felt guilty.

## **4. Stay On Target**

Emerson notwithstanding, inconsistency is a dangerous thing when exhibited by management.

### **4.1. Unmix Your Messages**

Corporate vision statements tend to be vague to the point of uselessness, but it is important to be clear about what's important and to be consistent about it.

- The Electronic Entertainment Expo (E3) is the big orgasmic event of the game industry, and thus, everyone in the industry expects to go every year. I know of one game company president who was unhappy with the progress of their game and told everyone they wouldn't have the usual company days off to attend. Then she changed her mind and was upset when one of the senior staff elected not to attend, anyway. So for future reference, is it important to attend E3 or not?

### **4.2. Use the Rhythm Method**

Ideally, a project should run like it's on autopilot. Everyone by default should know what to do every day (if you have to have a meeting every morning to tell people what to do, then certainly this is not the case). When you're running a marathon, you don't have to tell your feet to take every step - it's muscle memory.

- For example, builds ideally should be automated and run overnight, with the results emailed to everyone on the team. On one game project, I manually ran the build every morning and prepared it for QA, with only a handful of hiccups for an entire year, but still toward the end of the project people kept asking me if there was going to be a new build that day. So the last couple of months I ended up manually emailing build notices

every morning, also. I would have saved myself some grief if I had figured out how to automate the process earlier in the development cycle. That might have helped me avoid the problem of management asking me for new builds at various points during the day, which was not only annoying (imagine asking for a paycheck whenever you felt you needed one), but kept me from doing other work.

### **4.3. Stick with the Plan**

If you have a reasonable plan, stay with it. It's hard enough to get everyone on the same page in the first place. Tweaking your plan or worse yet, changing directions in midstream, will impose some overhead and delay from the "context switch" and lower confidence in the project.

- Shortly after I joined a video game project, my group was diverted from their planned tasks to throw together a demo disc. This was not surprisingly a messy patchwork process, but once it was nearly complete, management changed their minds and put us back on the original schedule, albeit one month behind. The demo disc was resuscitated a few months later, but by then it was old code nearly impossible to debug.

There are times when a plan is failing and changes are necessary to have any chance of completing the project. But in those cases, you should be able to identify clearly the points of failure and be able to justify that the next plan avoids those problems, and more importantly, avoids the pitfalls in reasoning that led to the first plan. Otherwise, there is no reason to believe that at some point during execution of the new plan that you won't change your mind again.

### **4.4. Stop Pulling the Fire Alarm**

Left to their own devices, management will often manufacture emergencies. Sometimes it's a conspiracy - one CEO will call a vendor CEO to complain and demand instant action. Or maybe Star Trek was on TV the other night, and the company president wants to be like Captain Picard ("Make it so!"). In either case, after raising some dust and getting some crunch time work, the bosses feel mightily pleased with themselves.

But an emergency indicates a failure in planning, and repeated emergencies will cause your staff to either leave or treat the emergencies as a joke (chances are you'll have both results - your more motivated staffers will leave and the less active ones will stay). Scheduled crunch times will have the same negative effects and add high risk to your project - if the crunch time is not as productive as expected, what are you going to do then? Double-crunch time? Try to understand - a smooth project is a successful project.

## **5. You're the Boss**

You're the boss, so there's no reason to pretend you aren't.

### **5.1. Don't Ask, Don't Tell**

It's not a democracy, so there's no point in soliciting input that you don't really want it.

- I worked for one manager who would ask everyone for their opinions and then get mad when someone disagreed with her. The only rationale I could see is that she would selectively bring up those cases when someone agreed with her to buttress her decisions later. But it would have saved a lot of time and argument if she had just made the decisions she was going to make anyway.

Of course, you shouldn't ignore good input from employees - just don't ask for an opinion if you don't want it. The most useless area to ask about is your own performance. It's like asking "Do these jeans make me look fat?"

- When I worked on the Hubble Space Telescope, I got into a dispute with a project lead who had made changes to my code when I was on vacation. He was unable to tell me exactly what those changes were, so I refused to integrate them. Offended, he asked me, "Don't you trust me to make those changes?" Now, why did he have to ask that?

### **5.2. Stop Whining**

Yes, being the boss is hard. But no one cares.

- One of my more pathologically self-absorbed bosses practiced management by self-pity: "I never would have started this company if I'd known how hard this is." "Do you want to trade places?" "I know your point of view, but you don't know mine." "I kept all my promises and everyone else broke theirs." Believe me, it gets old after a few months, not to mention several years. This person even expected her employees to feel sorry for her whenever she terminated one of them.

Everyone is guilty to of some rationalization and self-delusion, but people who actually want to boss others around tend to be guiltier than most. (Hint: if you spend a lot of time thinking about how you're a good guy, you're probably not)

- One abrasive company president often complained to me how she'd be treated differently if she was a man. A departing employee confirmed that, remarking "if she was a guy I would have punched her by now".

I enjoy working with managers who frankly discuss problems, but there's a difference between entertaining gripes and whining self-pity. If being in charge is really so unpleasant, quit and do something useful. If you feel unappreciated, tell it to your therapist. And if you're really narcissistic - well, you probably don't realize this section is about you.

### **5.3. Pick Something**

Real leadership involves making the tough decisions. If you can't make those decisions, then, as I rather bluntly told a manager once - what do we need you for? The first choices to make are in prioritizing, and everything has to be prioritized - features, bug fixes, delivery dates, customers, hiring, buying equipment, and so on.

- At one company where I was maintaining a Unix application while porting it to Windows, the president of the company told me the number one priority was to get the PC product out. When I asked him what about the Unix product, he thought a bit and said, "that's the highest priority, too". Thanks a lot, that's really useful.

Understandably, managers like to keep their options open. Sometimes this manifests as a refusal to make decisions early in the project, or a convenient amnesia regarding past decisions later in the project (you know this is going to happen when decisions are not written down). This strategy is as rewarding as setting up your retirement account just before your retire.

## **6. Lead By Example**

No matter how well-written the employee handbook or how well-produced the corporate video, it is management behavior that sets the tone of the corporate culture. "Do as I say, not as I do" works great if you're satisfied with a mediocre organization or if you rule by fear. But if you want a high-quality group and you're not Stalin, you have to lead by example:

### **6.1. Be Realistic**

The biggest management disease is wishful thinking.

- One Vader-like CEO intoned to me "This is the last time. You just have to take my word for it", when one of our helpful integration engineers insisted we had to deliver new features immediately after an official release. That assurance was as convincing as "this is my last drink". And it turned out to be just as valid.

Do the right thing now.

## **6.2. Be Responsible**

If there's one common skill among managers, it's self-promotion. And self-preservation (well, that's two, but they come as a package).

- When I worked on one defense simulation project that looked doomed, I noticed the head of the department stayed a healthy distance away from the lab, for several months. But just a week before the deadline, when it looked like we were actually going to pull this project off, he showed up in the lab glowing with enthusiasm and optimism.

If you expect your people to put themselves on the line, you have to stick your neck out a little, too.

- One of my more annoying bosses liked to go around telling people "I'm holding you responsible" but whined "Now everyone's going to blame me" whenever she screwed up.

Everyone makes mistakes. At least if you 'fess up, you'll have the respect of your employees, and maybe, just maybe, they'll take responsibility for their work, too. It may not be the best corporate politics, and maybe you'll have to adjust your self image, but it's good leadership.

- I worked for one director of engineering who readily admitted her mistakes although she could easily have blamed all problems on our dysfunctional software development group. When a rogue engineer pushed ahead with his own release (and his own agenda, to curry favor with the client) independently of the rest of the project, creating more work and more grief for everyone else, this director admitted almost immediately that it was a bad decision on her part to let him do it.

On another project where it turned out there was an embarrassing flaw in the demo release - the director of software could easily have blamed a number of parties, ranging from employees to vendors, but he simply stated that it was a collective failure, we had to focus on a solution and fix the process to avoid similar problems in the future.

If you make it clear you've learned from experience, maybe everyone else will. They'll follow you with more confidence, instead of fatalism. And if you accept responsibility for mistakes matter-of-factly, instead of in some torturous sequence of denial and blame, then maybe everyone else will do the same and get on with the real work.

- One of my bosses seemed to expect a Nobel Prize every time she admitted to a mistake. Months later, it would be the subject of lore - "Remember I took responsibility for that? Remember?"

I've seen some leadership advice saying you should never apologize. I don't really agree with that, but an insincere apology is worse than none at all.

- One of my favorites pseudo-apologies: "It's my fault for hiring that person." A feeble show of responsibility while directing the blame at someone else.

### **6.3. Be Professional**

Professionalism is underrated. Companies sometimes brag about how much fun they have and how they're one big happy family, but parties can only last so long, and how many families actually want to work together?

- I had one boss who liked to admonish her employees to "please be professional". Which is one of the more annoying things one can say, like "don't be an ass". It was also an ironic thing to say, since her management repertoire included whining, yelling, foot-stomping, tossing items on employee's desks and terminating conversations by abruptly turning around and leaving.

You shouldn't have to like all your coworkers to have a decent job. (That would really limit your options) As with politeness, professionalism can make bad situations tolerable and make good situations last longer.

- The aforementioned employer told me she didn't care if departing employees had resentful feelings about their jobs, and then in the same conversation she wondered how to keep ex-employees from spreading a bad reputation for her company. Apparently, she didn't see the link.

Typically, even inept and slimy employers will be smooth when easing employees out the door. "Glad to have you, sorry it didn't work out."

### **6.4. Be Fair**

Don't be fooled by seeing all those warm bodies in the cubicles staying late in the day - they're not necessarily working (the Internet is a wonderful thing), and there's sure to be a stampede for the exit once the boss's car leaves the parking lot.

But employees might conscientiously put in the effort if they see that effort from you.

- On one tightly-scheduled porting project, I was responsible for the bulk of the code and consequently spent the first few months working eighty-hours weeks. I didn't get much assistance from outside the group, but I was greatly appreciative when the other members witnessed my long hours and offered to help in any way they could.

In particular, if you expect people to go above and beyond the call of duty and give up their personal time, you'd better show you're willing to make the same sacrifice.

- I've been on more than one project where the manager, just before going to the gym, would tell everyone to stay late at work. One manager asked me to wait for her in the office after hours until her aerobics class finished - upon return she ignored me another half hour until I finally asked what she wanted to talk about. "I just wanted to tell you you're doing a good job." Wow, that was worth skipping dinner.

The message - my time is more important than yours. Now get back to work.