

CHAPTER 2:
POLITICAL PITFALLS

...there is nothing more difficult to carry out, nor more doubtful of success, nor more dangerous to handle, than to initiate a new order of things. For the reformer has enemies in all those who profit by the old order, and only lukewarm defenders in all those who would profit by the new order, this lukewarmness arising partly from fear of their adversaries...and partly from the incredulity of mankind, who do not truly believe in anything new until they have had actual experience of it.

—Niccolo Machiavelli, *The Prince*

Political pitfalls are perhaps the most treacherous, because they have little to do with skill, technology, or the worthwhile nature of a given project. Instead, they have to do with credit, blame, power, control, promotion, personalities, gossip, image, past slights, called favors, sabotage, championing, and tradition. Developers and other technical types are prone to blunder into political pitfalls because developers like to think of themselves as rational technocrats and assume that others will think as they do and be motivated as they are.

At the same time, it is a common error on the part of upper management to think that developers and other technical types are naive or ignorant about organizational politics. Developers are often very astute and aware of politics, but they find it disgusting and destructive (rightfully so) and tend to dismiss it as not worth considering (wrongly so). This tends to play into the hands of upper management, who feel that they have everything well in hand. They don't realize the real danger they face from a group of ticked-off developers who decide to "hack" politics: that is, to use (and abuse) organizational politics to achieve an end. Many mid-level managers (and some high-level ones) have found their power base eroded and have even found themselves out of a job because of developer sabotage or rebellion, whether overt or covert.

The point is that organizational politics does exist, it is significant, and it can be ignored only at the peril of your project and possibly your job. The pitfalls listed here are those you are most likely to fall into when attempting to initiate or promote the use of object-oriented development within your organization.

Pitfall 2.1: Not educating and enlisting management before the fact.

There is an oft-cited dictum in technology development groups: “It is easier to beg forgiveness than ask permission.” In fact, it was a motto of pioneer programmer Admiral Grace Hopper. It is often true and sometimes crucial to circumvent bureaucratic foot-dragging and politics. But it is not always the best course, and the danger of following it is commensurate with the technical, financial, and political risks involved. And all three risks abound when an organization is moving to object-oriented development for the first time.

Nevertheless, it is not uncommon for a development group to make the switch to OOD with only minimal involvement and education of upper management. Indeed, management may not care very much or may even be supportive—having read an article about the wonders of object technology in a weekly business magazine.

And that is where the pitfall lies. For unless the project comes in within acceptable time and budget limits, upper management will suddenly be asking hard questions reflecting reasonable or unreasonable expectations, given what they know or were told.

Symptoms	Apparent lack of knowledge or overly positive expectations or both on the part of management about OOD and its role in the relevant projects. Sudden distancing or self-protecting activities if problems crop up.
Consequences	Lack of support and possibly active hostility from upper management if problems arise or if their expectations aren't met. Finding yourself twisting in the wind. Career damage, lack of promotion, demotion, loss of job.
Detection	Sit down with upper management and find out how much they understand about object technology and how it's being applied in the relevant project(s). Have them detail their expectations. Find out their level of enthusiasm or concern for the use of OOD.
Extraction	There's little to do except start the education and enrollment process much later than it should have been. It may be really tough, depending on how current expectations compare to reality, and the truth is, there's no good reason for not having done this before things got started.
Prevention	<p>From the pitfall itself, it's obvious that there are two separate (if related) tasks: educating and enlisting. The first comprises letting management know exactly what OOD entails, which means that you had better know yourself, and you'd better know it well enough to explain it to nontechnical people. Work to contain your own enthusiasm for OOD; remember that it is safer to underpromise and overdeliver.</p> <p>Second, and this is probably tougher, you need to enlist key people: those who can affect your budget and resources, and those who can affect the scope and direction of your project. If you can, prove yourself with small projects to establish credibility and to give your sponsors a track record to point to (and take credit for).</p>

Pitfall 2.2: Underestimating the resistance.

Objects are wonderful. At least, *you* think they are, based on anything from a breathless magazine article to years of experience with solid, successful object-oriented development. Or you may not think they're wonderful, but you do think they may offer advantages and benefits to your development efforts. And, of course, what's obvious to you should be obvious—or, at least, understandable—to everyone else.

So you blithely push ahead...until people start pushing back. And before you know it, you're engaged in a political battle, fighting resistance to object technology that may range from guerrilla sniping to a full frontal assault.

Why might people resist accepting object technology? The reasons are varied and may range from the well-founded to the completely irrational to the deliberately obstructionist. Here are examples:

- They don't understand object technology.
- They don't want to understand object technology.
- They're afraid they won't be able to understand object technology (and thus will have less value to the company).
- They know they won't be able to understand object technology, because they've been sloughing for the last ten years and know that this will expose them for what they are.
- They have very strong feelings about the language and/or programming methodology, or tools or all three that they currently use.
- They really detest the language, methodology, or tools or all three that you're proposing they use.
- They're worried that the reasons for adopting object technology aren't well thought through.
- They're worried that the plan for adopting object technology has significant problems or flaws.
- They want to adopt object technology, but they want to do it their way.
- They want you to fail so that they (or someone they like better) can get your job.
- They read this book.

Compounding the situation is the fact that you may face several of the reasons simultaneously, possibly from different sources.

Symptoms Delays in getting approval or support. Rumors circulating behind your back. Objections constantly brought up in meetings. Former supporters distancing themselves from you. Reduction in project priority and resources.

Consequences Significant project delays or project failure. Loss of influence. Loss of job.

Detection If you suspect that resistance is deeper or more entrenched than you thought, you need to find out the sources of resistance and the reasons behind it. This can be hard, because the people resisting you may not be honest about what they're doing, or why they're doing it.

Furthermore, if the resistance is coming from above, your boss(es) may feel no obligation to explain their reasons.

Extraction You have four choices, not necessarily mutually exclusive, based on the nature, depth, and source of the resistance:

- Push ahead in spite of it.

Pitfalls of Object-Oriented Development by Bruce F. Webster (original edition)

- Mollify or enlist those who resist.
- Redirect the project to quell the objections.
- Abandon the project gracefully and (maybe) try again later.

Prevention

Make a list of everyone who might have any input or influence and judge where they stand. If possible, meet with each one privately to sound out her or him—but recognize that you may not get an honest answer. Offer each person a chance to make criticisms and recommendations; let everyone feel a part of the project and the process. Build enthusiasm, pointing out specific benefits, particularly those of interest to each individual. Use each source to cross-check others. This process is known as “getting your ducks in a row,” and you’d better do that before you start. This process may be more critical for those below you than those above you; don’t underestimate the power of developers to make your life wonderful or miserable (see this chapter’s introduction as well as Pitfall 3.2).

Having done that, assess the costs and risks in pushing this project forward compared to the possible benefits and rewards. Factor that with the probability of success and make your call. If it looks too dangerous, scale back or redirect. If it looks impossible, find somewhere else to work.

Pitfall 2.3: Overselling the technology.

It's easy to get excited about object-oriented development. It works very well for small-scale projects. It has real benefits for managing complexity. It builds very well on what we've learned about software engineering over the past 25 years. It is an important—and perhaps an essential—part of solving the growing software development crisis.

Enthusiasm for object technology, particularly for individuals relatively new at it or with a vested interest in its adoption, may lead them to give glowing descriptions of its wonders and benefits to you and others.

Object proponents may make these comments in all sincerity and innocence. Or they may make them with the knowledge that they are stretching the truth a bit or perhaps rupturing it altogether. But whether they know that what they're saying is accurate is, to a point, immaterial: When reality intrudes, the results will be the same, and their lack of knowledge or their lack of accuracy will convict them equally.

Symptoms	There are two phases. First, those who have been oversold express expectations of the object technology that make you increasingly uncomfortable; you may find yourself starting to downplay things a bit. Second, as problems start to pile up, there are growing questions, doubts, and expressions of mistrust and cynicism.
Consequences	Those who trusted what they were told will feel betrayed; those who did not will feel justified. In either case, both the reputation and the influence of those doing the overselling will shrink, and it will be a long, hard road to recovery.
Detection	Sit down with a sheet of paper and write every promise, hope, and expectation that you or others have made in the name of object-oriented development. Ask yourself how supportable or valid each is. If you have questions about your ability to judge this, find someone who knows more about OOD than you do and ask them for help evaluating how supportable or valid each point is. Ask yourself whether you've succumbed to wishful thinking.
Extraction	If you or others have oversold the technology, you need to start readjusting expectations immediately. It is probably best to do it in one massive reset than to do it piecemeal—a single sword thrust is, in the end, less painful than a thousand paper cuts. But it's going to take a long time to reestablish your credibility.
Prevention	Several times in this book you will find the recommendation to underpromise and overdeliver. Believe it and make sure that others believe it. If you have questions about your ability to judge, then bring in two or three experts to consult and weigh their opinions. On the other hand, if you knowingly oversell, telling yourself that you can manage things down the road, you will almost certainly be wrong. Resist that temptation. It will always get you into trouble, and your own integrity will be diminished. It's just not worth it.

Pitfall 2.4: Getting religious about object-oriented development.

Let's try to keep our perspective while standing knee-deep in the hoopla. As anyone experienced in this area will tell you, object-oriented development is not going to end world hunger or bring about peace in our time. It won't transform the country of your choice into the new economic superpower of the 21st century. It won't end cancer, reduce the number of handgun deaths, or curb the rate of out-of-wedlock births. It won't even affect dandruff.

Closer to home, OOD doesn't invalidate existing programming languages, methodologies, and tools. It won't make good engineers or architects out of bad ones. It also won't make good products out of bad ones; although it may allow you to write a better program than you might have otherwise, you can write awful programs just as well using object technology as you can using, say, BASIC or Cobol or your favorite scapegoat.

Switching to object technology probably won't make a late project ship on time, though it may allow you to complete a project that never would have shipped otherwise. On the other hand, it may make the project even later. OOD won't have a major impact on the time and effort required for analysis, design, and testing—except, possibly, to increase it. You won't be able to reuse 90 percent (or 60 percent or probably even 30 percent or maybe even 10 percent) of the code from your first project. OOD may not even be the best solution to a given problem. It might, in fact, make things worse, not better.

Symptoms	An almost blind faith in the virtues of OOD and an equal blindness to its pitfalls and failings.
Consequences	Arguments with others; a lack of flexibility; blindness to weak spots.
Detection	Analyze reactions to each of the statements above. When people disagree, note any strong negative emotional response. Have those who disagree with a given statement explain it to a disinterested, yet knowledgeable third party and see whether they think the logic holds water.
Extraction	As with many pitfalls, study and experience will do much to cure this one. Beyond that, the Detection method above will help you to identify specific areas of excess enthusiasm and credulity.
Prevention	Read the books in the Bibliography. Find people who have completed meaningful projects using OOD and ask them for their lessons learned. Have them read the opening paragraphs of this pitfall and respond to any items they disagree with.

Pitfall 2.5: Not recognizing the politics of architecture.

Once, while discussing the challenges of object-oriented development with Taligent trainer Tom Affinito, I mentioned—citing Fred Brooks—the need for a chief architect (see Pitfalls 4.13 and 4.14). Tom immediately responded, “Yes, and ultimately architecture is a political act.”

Unless you are both sole architect and sole implementor—and maybe even then—architecture is a political act, and it is especially true in development groups of any size. Developers like to think themselves above politics. Not so; they are as human as anyone else. Developers can be quite political and will use a variety of techniques, good and bad, to achieve their ends: persuasion, hyperbole, fear-mongering, caucusing, lobbying, consensus building, rumors, backbiting, leadership, intimidation, sacrifice, tantrums, threats, and subversion, to name a few. The fact that developers are generally very bright just increases the effectiveness of whatever techniques they use.

Architecture is the lodestar of developer politics because it is ultimately the most prestigious role in software development. Brooks goes to great lengths (and rightfully so) to assert the intellectual and creative challenges of implementation, but it is Frank Lloyd Wright’s name that remains attached to the structures he designed, not the name of the contractors who built them. Or, to update an old programming joke, the verb “to develop” is conjugated this way: “I architect, you implement, he tests.”

Yet we may fail to recognize or acknowledge this situation and its implications. Why? First, we want to believe that all developers have (with regard to the project) the same intentions and goals. Second, we want to believe that the best course is the one that is obvious to us and that everyone else will agree with that. Third, we don’t want to have to deal with politics ourselves, particularly politics that can lead to confrontations.

Symptoms Assuming that all other developers have the same goals, ideas, and talents. Not wanting to assert architectural issues. Finger-pointing as design problems arise.

Consequences Hidden or overt team dissension, leading to poor morale and lack of cooperation as well as a weakened architecture.

Detection Ask yourself these questions:

- Who is the chief architect of this project?
- How well do the other developers support the chief architect?
- How effective is the chief architect in her role?
- What are the obstacles she faces?
- What are all the political issues involved?

Answering these questions should go a long way toward determining whether you have underestimated the politics of architecture.

Extraction The first issue to address is that of authority commensurate with responsibility: If there is a chief architect on the team, does she really have the authority to compel adherence to the architecture? This kind of authority is different than being able to say, “Do this or you’re off the project,” which often leads to subtle or overt rebellion by team members. This kind of authority accrues from several factors:

- a proven track record
- respect from teammates

Pitfalls of Object-Oriented Development by Bruce F. Webster (original edition)

- support from technical and upper management
- a mutual pact with the developers that the chief architect will give serious consideration to all their ideas and suggestions but that they will abide by her decisions
- a willingness to give in on the small things so that she can hold her ground on the big things

Beyond that, healing a team that is experiencing jealousy and dissension is no easy feat, especially when you're in the middle of a project. Books and seminars on team-building abound; study, learn, and apply.

Prevention Go through the questions in *Detection* but cast them in the future tense. Then, as with *Extraction*, proactively work to build the team and establish the authority of the chief architect.

Pitfall 2.6: Getting on the feature-release treadmill.

You know the drill. By hook or crook, through long weeks and late hours and ruthless compromising, you finally deliver the project. It's finished, it's out the door, and you have taken a few weeks to remind yourself what real life is like. Now you have the opportunity to go back and make right all the hacks and shortcuts and ugly patches you used to get version 1.0 of the project to the customers. Besides, having done the project once, you now have a far better idea of how it should have been done in the first place. You rub your hands together and...

...you are handed a list of new features required by customers or potential customers for version 2.0 of the project, which has to ship far sooner than you would have thought. After some study, you realize that you're not going to be able to deliver all those features within the required time frame—and even as you realize that, more features get handed down.

Your plans for cleaning up the code and the architecture are rapidly getting pushed off to version 3.0, and you really wonder whether things will be any different then.

Chances are, they won't.

Why does this happen so often? There are several reasons, really. One may be economic necessity. Your company (division, group) may have to fund itself, which means that sales have to be sufficient to meet all expenses, including your salary and benefits. This is especially true in the lean and mean decade of the 1990s, when, as one wag has put it, the new status symbol is a job. If that's your situation, then don't worry as much about this pitfall; architectural purity is nice, but a paycheck is better.

Another, less acceptable reason may be a short-term focus on the part of upper management. Determined to milk a current market opportunity and concerned primarily about next quarter's results, they may not be receptive to an engineering investment that would yield more in the long run at the expense of smaller profits right now.

A third reason—not necessarily exclusive of the other two, and sometimes justified—is a suspicion on the part of upper management that engineers are more interested in doing something “cool” or “elegant” than in doing something profitable. What upper management may not understand is that elegance—architecture and code that are concise, yet clear and comprehensive, tending toward orthogonality—always pays off. Always. The only question is how soon and how much, and that is the fulcrum for the balancing act of the technical manager. Ironically, when the engineering staff is able to rapidly deliver the features requested by upper management, it is often because of a previous investment in elegance; likewise, when features take a long time to implement or bugs take a long time to fix, it's often because of architectural gaps and past short-cuts.

Symptoms	Constantly having to push engineering work into the next planned release and not the current one.
Consequences	The cost of adding new features or extending current ones goes up, not down, over time. The program gets larger, less stable, and less cohesive. Bugs multiply, possibly causing a customer backlash.
Detection	Sit down with upper management and detail the engineering work that needs to be done. Indicate what impact this will have on the current list of desired features. See what kind of reaction you get.

Pitfalls of Object-Oriented Development by Bruce F. Webster (original edition)

- Extraction** It ain't easy. The reaction you get under Detection should give you a pretty good idea of what things will be like. There are two approaches, which should probably be used together. First, actively work with upper management and customers to determine which features are absolutely necessary and which are merely desirable —ones they could live without until the next feature release. It's critical to talk directly with the customers, because they may not have done their own prioritizing; the list you get from management may reflect everything from essential requirements to blue-sky wishes.
- Second, build engineering cleanup time into each feature so that an engineer allocated four weeks for a given feature spends, say, three weeks on the feature and one week on architecture.
- Prevention** Besides using the steps described above in Extraction, you need to sell the legitimate, long-term benefits of object-oriented development and emphasize the need to make the proper ongoing engineering investment in order to get those benefits. This means investing the time to educate management and customers, to set proper expectations, and (if possible) to run a trial project to demonstrate the process.
-

Pitfall 2.7: Betting the company on objects.

Imagine the following scene. Your company's executive staff gathers for a presentation on a new technology that will revolutionize information productivity. After a presentation citing the ongoing problems of information management, enterprise computing, and competitive response, you are presented with the solution that will boost the company's bottom line and guarantee its future: structured development!

But wait! you say. Structured development has been around for a long time. Lots of folks use structured development and have mixed results; indeed, most don't use it well. Lots of companies have gone out of business while relying upon structured development. How is structured development going to guarantee our future?

Good question. And yet, structured development and its associated methodologies are more mature, established, and well understood in real-world applications than are object-oriented development and its methodologies. For that matter, the number of competent, practicing engineers who are expert at structured development is vastly greater than the number of competent, practicing engineers who are expert at object-oriented development. If a company can't succeed using structured development, why does it think it can succeed using OOD?

You may think that I'm being reactionary or that I don't understand all the benefits OOD has over structured development. If so, you miss the point: It is not *technique* (using Jacques Ellul's term) in and of itself that will benefit the company, but rather its intelligent and purposeful application by people who know what they're doing and why. To think that adoption of object technology will suddenly make everything better is irrational to the point of superstition.

Symptoms	Unrealistic expectations on the part of upper management. Marketers overpromising delivery times and feature lists. Technical managers who see object technology as a silver bullet. Developers neglecting solid software engineering practices, because "objects don't require them."
Consequences	At best, you go through a wrenching reeducation and attitude adjustment. At worst, you lose the bet.
Detection	Take the company's current business plan, mission statement, and product plans, and eliminate all references to and consideration of object technology. Do all of these documents still all make sense?
Extraction	Point out, repeatedly, that it doesn't matter what technology you use if the products aren't worthwhile and if the company can't get a return on investment. Enumerate the factors required for success independent of object technology. Look at all that you can do to maximize those factors. Then—and only then—look at how object-oriented development can aid and support those efforts.
Prevention	Bet the company on people, not on objects. Isn't that what politics is all about?

Conclusion

The term *politics*, like *politician*, has gained a patina—maybe a crust—of oil and mud. The very term carries a reek of manipulation, dissembling, power seeking, battle lines, patronage, and self-aggrandizement at the expense of others. I suspect that more companies falter and fail due to the consequences of internal politics than for any other reason.

But the obverse of that coin has writ on it leadership, consensus building, mutual loyalty, team spirit, and self-sacrifice for the common good. Cynics smirk at such concepts in a business setting, deriding them as naive at best and deceptive manipulation at worst. Yet these impulses are as valid as those above and are more likely to lead to success.

It is a sad commentary on human nature that office politics partake too much of the former and not enough of the latter. But it is a true observation, and so be aware of and beware of the political pitfalls that abound.

References

Ellul, Jacques. *The Technological Society*. New York: Vintage Books, 1964.

Machiavelli, Niccolo. *The Prince* and *The Discourses*. New York: Random House, 1950. (Translated by John Wilkinson).