3 Negotiations

Only free men can negotiate. Prisoners cannot enter into contracts.

—Nelson Mandela
Statement from prison, Feb. 10, 1985,
refusing the terms offered for his release
by South African President P.W. Botha
Quoted in Higher than Hope*, Part 4, Chapter 30, Fatima Meer (1988)*

If you're the manager of a death march project, it's very easy to predict the outcome of negotiations over budget, schedule, and resources: *You lose*. This is almost inevitable, because such negotiations take place at the beginning of the project (or even before the project is formally initiated), when the project owner/customer has neither the intellectual ability, the emotional stamina, nor the political necessity to accept the unpleasant counter-offers of the project manager. More rational negotiations sometimes take place a month or two before the deadline, when the first project manager has quit or been fired, and when a new project manager demands (as a condition of accepting the assignment) that everyone face up to the reality that the original deadline, budget, and required functionality will never be achieved.

But none of us seems willing to accept this sad state of affairs. Thus, even though this chapter probably *could* focus on rational negotiating strategies for the replacement project manager, I'll nevertheless confront the question most of us wrestle with: How can we negotiate a tolerable set of conditions at the *beginning* of a death march project? Alas, there are no magic secrets to be revealed in this chapter; the sad reality is that at the end of the process, you lose. Still, it's useful to be aware of the devious political games by which you're likely to be outmaneuvered, as well as the options that should be explored when you have been presented with a completely unrealistic schedule, budget, and/or staffing constraint.

My assumption throughout this chapter is that *you* are the one involved in the negotiations about death march projects, schedules, and so on. If you're a technical staff member, you may be indirectly involved—for example, providing advice and estimating data to the project manager, so that he or she can carry out the negotiating battles with higher levels of management. But in an email communication with Doug Scott[1] recently, I was reminded that in some projects, even the project manager has only an indirect role, because all of the negotiations are being made on his or her behalf by the next higher up manager:

> *...my biggest single obstacle in deathwatch projects has been my own management. I came to the UK in 1972, and moved on to big projects almost immediately. I don't think I learnt anything about running projects since that date (I learnt a lot about politics, but that's*

something else). You need to understand your own management's negotiating stance, and if they love to play roll over, you have to keep them well away from the project.

RATIONAL NEGOTIATIONS

The suggestion that we really *do* know how to accurately estimate the required schedule, budget, and resources for a nontrivial project will set off an emotional debate among any group of software professionals and managers. Our track record over the years certainly hasn't been a very good one; on the other hand, many would argue that the problems have been the result of political games associated with the very death march projects that we're discussing in this book.

But most large organizations can point to dozens of projects where the software team made its own schedule, proposed its own budget, and expressed supreme confidence that it would deliver a fully functional system within those constraints; the team then proceeds to hoist itself on its own petard and fails to deliver anything, any time. So it's no wonder that in many of these organizations, the user community and senior management have given up on the negotiating process and have instead begun imposing "do-or-die" deadlines and budgets. Such is the genesis of many a death march project.

Still, that doesn't mean that we should abandon all efforts to derive a "rational" estimate that we can use in the preliminary negotiations for a project. Indeed, it's crucial that the project manager avoid the temptation to give up and simply accept the initial death march project constraint as an edict. One of the common signs that a project team has adopted what I called a "suicide-style" behavior in Chapter 2 is the attitude—expressed by the project manager and echoed by the team members—that "we have no idea how long this project will really take, and it doesn't matter since they've already told us the deadline. So we'll just work seven days a week, 24 hours a day, until we drop from exhaustion. They can whip us and beat us, but we can't do any more than that..."

I'm not going to discuss estimating techniques at length in this book; if the project manager has no skill or experience in estimating, then a death march project is no place to begin learning. But let me point out some of the obvious resources that we have available in this field:

- *Commercial estimating tools*—Products such as SLIM and CHECKPOINT are available from Quantitative Systems Management and Software Productivity Research (SPR). SPR's Chairman, metrics guru Capers Jones, estimates that there are some 50 commercial project estimating tools. None of them are perfect, and all of them require intelligence on the part of the user (garbage-in/garbage-out applies in this field, too!), but in the best case, they can produce estimates that are accurate to within ± 10%. Even if they're only accurate to within ±50%, it's better than the political demands that the project manager is coping with, which are often 1,000% beyond the ability of the team to deliver.

- *System dynamics models*—Numerous simulation models have been developed to explore the nonlinear interactions between various factors that affect a project's behavior. For example, if part of the strategy of a death march project is to impose a demand for heavy overtime on the part of the project manager, what will be the effects over a period of weeks or months? The natural assumption is that more "output" will be produced than would be the case with a normal eight-hour working day, but most experienced project managers will also point out that productivity (measured in function-points per day, or lines of code per hour, etc.) gradually decreases as exhaustion builds up. Error rates also begin to increase, which have an obvious impact on testing and debugging effort. And if the overtime continues long enough, the project team eventually collapses from exhaustion. Of the simulation models that I've seen in this area, the best is Tarek Abdel-Hamid's [1], which has been implemented in languages such as DYNAMO and iThink.
- Dozens of articles and books have been written on the topic of project estimating. Barry Boehm's *Software Engineering Economics* [2] is a good place to begin; it's important to note that Boehm's COCOMO model from the early 1980s has been updated to COCOMO-2 [3]. Another classic is Brooks' *The Mythical Man-Month* [4]; this has also been updated recently to reflect modern technology and software practices. More recent books on software estimating include Jim McCarthy's *Dynamics of Systems Development* [5].
- The *process* of estimation has been studied and documented, and organizations like the Software Engineering Institute have published useful guidelines and checklists for improving the process of estimation [6,7]. Even if we aren't very good at it, we know how to get better.
- Familiar techniques such as prototyping and time-boxing can be used to get an accurate picture of how feasible or infeasible the project constraints are for the overall system being developed. This is by no means a foolproof approach, but it can inject a dose of reality into the project team and the surrounding layers of managers and customers. If management is demanding a system that will require a team of three to write a million lines of code in 12 months, then it should be possible to define a skeleton version of the system that can be built within the first month; this will provide at least a rough calibration of the team's level of productivity, as well as a rough idea of the overall feasibility of the project.

IDENTIFYING ACCEPTABLE TRADEOFFS

Let's assume that the project team has prepared a "rational" estimate of the schedule, budget, and personnel required for a death march project; and let's assume that management is prepared for some kind of give-and-take process of negotiation before the final decisions are made. The most common situation is that management will declare the initial estimates "unacceptable" and make counter-demands that are far more stringent. What should the project manager do?

As author/consultant John Boddie[2] pointed out to me in a recent email message to me, the crucial thing is to ensure that everyone agrees that there is more than one possible "scenario" for the project:

> *Some useful questions during negotiations,*
>
> *"If the system is ready on the fifth of September rather than on the first, will we already have declared bankruptcy September second?"*
>
> *"Is there an 80/20 rule here? If we deliver the critical 20 percent that gives eighty percent of the value, do we need the twenty percent at initial roll-out?"*
>
> *"Everybody wants things good, wants them fast, and wants them cheap. Everyone knows that you can actually achieve any two of the three. Which two do you want?"*
>
> *The principle at work is to make those who are demanding the death march look unreasonable if they are unwilling to consider more than one possible outcome. Unless there is an acceptance of more than one way to approach the problem, then there is no negotiation. All the manager can say is, "We'll give it our best shot, but there are no guarantees."*

If the counter-proposal from senior management or the customer involves only one "variable," then the project manager can estimate the impact on the other variables. For example, if the manager's first estimate is that the project will take 12 months with three people and a budget of $200,000, it's possible that senior management's first response will be, "Baloney! We need to have that system up and running in *six* months!" The obvious way to accomplish this is to add more people and/or spend more money (e.g., to pay higher salaries to hire more productive programmers).

But Fred Brooks told us nearly 30 years ago that the relationship between time and people on a software project is not a linear one; the term "man-month" (which would probably be expressed as "person-month" in today's politically correct organizations) was thus exposed as a myth. Indeed, the relationship between *all* of the key variables in a project is likely to be nonlinear, and it's likely to be time-sensitive as well: Because of the "feedback effect" of many management decisions, a change in one variable (such as adding more staff) will not only have an impact on other variables (such as productivity) over time, but will eventually have an impact on the original variable—for example, the hiring of additional staff could lower morale, which raises the turnover rate within the project, which ultimately reduces the size of the staff.

The nonlinear, time-sensitive nature of these interactions is the essence of the systems dynamics models mentioned above; but it's also the reason for using the various commercial estimating tools described earlier. There is a key point here: The mathematics behind the systems dynamics models is typically based on nonlinear differential equations, and most of us aren't very good at doing that level of mathematics in our heads. Similarly, the commercial estimating tools carry out elaborate calculations involving dozens of parameters; trying to do this intuitively, based on a "gut" feeling for the situation, is likely to be quite error-prone.

Unfortunately, that's exactly the situation many death march project managers find themselves in. Sometimes this is because of the nature of the negotiating process (particularly a game called "Spanish Inquisition," which I'll discuss below); but it's also caused by the lack of estimating tools and expertise in many organizations. Again, this is not a problem you're going to be able to solve in a death march project if it hasn't been addressed already: If the organization is accustomed to deriving its project estimates by scribbling numbers on the back of an envelope, the death march project manager probably won't get away with spending $10,000 on a sophisticated estimating tool.

So what should the manager do in a situation like this? In the extreme case, the manager should recognize the futility of the situation and respond appropriately; I'll discuss that in more detail in a section below. But in the less extreme case, here are two guidelines:

- If the negotiating demand from users or senior management involves a change of less than 10% in one project variable, then you can compensate by increasing one of the other variables in a direct, proportional fashion. Thus, if management wants the schedule reduced by 10%, then add 10% to the size of the project team. This isn't entirely accurate, but it's a good first-cut approximation, and is about all you'll be able to get away with from a negotiating perspective.

- If the change involves more than 10% in one dimension, then you should assume it will have an "inverse square law" impact on any other single dimension. Thus, in the scenario above, management wants to reduce the project schedule from 12 months to six. Rather than responding by doubling the size of the project team, the manager should *quadruple* the team—or quadruple the budget, in order to hire super-programmers who can code with both hands at the same time. Without a formal estimating model, there's no way to know whether this crude heuristic will be accurate for any specific situation, but at least it's better than falling into the trap or negotiating a "linear" exchange of time for people. Unfortunately, the inverse square law is difficult to negotiate, and there's a good chance that the project manager's "outrageous" demands will be beaten down; but with luck, the manager will still end up in a better position than with a linear exchange.

NEGOTIATING GAMES

Negotiating *is* a game, and it takes place on all software projects; what's different about death march negotiations is that the stakes are much higher, emotions are much more highly charged, and the demands of the other side (in terms of schedule, budget, etc.) are usually so extreme that they overwhelm any "safety factor" that we might have used in the past. The most obvious safety factor in a traditional project, for example, is overtime: even if the project manager has been browbeaten into a tight schedule and restricted budget, success can still be achieved by asking the project team to work 10-20 hours per week of overtime for the final few months of the project. The additional effort doesn't show up in the official records, because the programmers aren't paid for overtime work; thus the manager ends up looking like a hero.

But in a death march project, modest amounts of overtime are typically inadequate to achieve the dramatic results that are being demanded. Besides, the users and senior management aren't naive: they *know* that overtime effort can be requested, and they've factored that into their own estimate of the "required" schedule for the project—thus pre-empting the manager's opportunity to hide that free resource. But project managers who are veterans of such negotiations have a few tricks up their sleeves, and the bargaining sessions begin.

The neophyte project manager is at a terrible disadvantage; in the extreme case, the neophyte isn't even aware that his past successes may have occurred *only* because the project team voluntarily contributed sufficient overtime effort to compensate for a ridiculous project schedule. And the ridiculous schedule may have been imposed upon the team precisely because of the manager's naiveté in the area of estimating negotiations.

Management consultant Rob Thomsett has described the most common negotiating games in a wonderful article [8]; I've summarized the more familiar games below:

- *Doubling and add some*—This is a ploy that has been used on projects dating back to the Pyramids, if not earlier. Use whatever estimating techniques you have available, then double the "rational" estimate; for added safety, add three months (or three weeks, or three years, depending on the overall size of the project). The major problem with this strategy is that it runs head-on into the most pressing constraint associated with death march projects: schedule compression.
- *Reverse doubling*—As noted earlier, management hasn't been oblivious as software project managers have attempted to "pad" their estimates by the doubling strategy discussed above. One reason for this political astuteness is that the senior managers in many organizations today are former IS/IT project managers—so they're intimately familiar with the games involved. As a result, they take the initial estimate given to them by the project manager and automatically cut it in half. Pity the poor neophyte project manager who didn't realize that he was supposed to double his estimate at the outset!
- *Guess the number I'm thinking of*—This is a game I learned in one of my first projects as a junior programmer. The user or senior manager has an "acceptable" figure for the schedule, budget, and/or other aspects of the negotiation, *but refuses to articulate it*. When the project manager offers his estimate of schedule and budget, the user/senior-manager simply shakes his head and says, "No." The implied message is, "That's too much. Guess again." The hapless project manager eventually (sometimes after half a dozen attempts!) comes up with an acceptable estimate, but because it's *his* estimate, the user/senior-manager is all the more determined to hold him accountable.
- *Double Dummy Spit*—"Dummy" is Australian slang for a baby's pacifier, and "spit the dummy" is an Australian phrase describing a baby so frustrated and angry that it spits out its pacifier. Thomsett uses this as a metaphor to describe the negotiating sessions when a senior manager erupts in a fit of rage when the project manager first makes his proposal for the death march project schedule and budget. The chastened manager scurries away, comes back with a revised estimate, and the manager erupts again—hence the "double dummy spit." The idea is to get the manager so cowed and terrified that he'll go along with anything in order to avoid yet another temper tantrum.

- *Spanish Inquisition*—This occurs when the project manager walks into a meeting of higher level managers, completely unaware that he is going to be asked to make an "instant estimate" for the death march project. Imagine a roomful of grouchy vice presidents staring at you while the CEO asks you in thunderous tone, "So, Smithers, when do you expect to get the Frozzle system done? I've told the whole management team that we'll have it online by March 13th of this year—you're not going to let me down, are you?" If you're brave enough to suggest that November 13th of *next* year would be a more realistic estimate, you'll have a dozen inquisitors questioning your intellect, your credentials, your loyalty, and perhaps even your religious faith.
- *Low bid*—With outsourcing a option in many organizations today, this game is becoming more and more common; it's also common in any situation where a software development organization is bidding against other competitors for the privilege of developing a system for a client organization. The game is obvious: The customer (or sometimes the development organization's marketing representative) tells the project manager that one of the other bidders has proposed a faster development schedule and/or a lower budget. This puts pressure on the project manager to not only match the competing bid (which may or may not be a real bid), but also to improve upon it in order to raise the chances of getting the contract. A variation on this game occurs when the client lets it be known that he's considering the option of not doing the project at all; a software development organization that's desperate to get the approval to initiate the project (perhaps because it will advance the career of the IS/IT vice president) will ensure that the project proposal is so attractive that it will be approved. Of course, this means that in many cases, one or more members of the IS/IT hierarchy *knows* that the project proposal is unrealistically optimistic and perhaps even a blatant lie. This in turn leads to the "gotcha" and "Chinese water torture" games described below.
- *Gotcha*—The "gotcha" game is sometimes played by the project manager as a way of getting revenge: Though he knows at the outset that the project proposal in unrealistic, he accepts it anyway, relying on the probability that by the time everyone is forced to face up to reality (e.g., a week before the deadline), it will be too late for the client to back out. But it's a dangerous game, because the client has to ask himself whether he wants to throw good money after bad; if the organization has a track record of previous projects running amok in this fashion, the client may decide to cancel the project and write off the expenses as a bad investment. But the chances are that the death march project *won't* be canceled right away because it's usually associated with business objectives, legal requirements, or political

battles that are difficult to walk away from. But that doesn't prevent the customer from seeking revenge for having the game played on him, and the most obvious form of revenge is to fire the project manager. This is also a common political ploy for various higher level managers and marketing representatives (who may have been responsible for the death march project commitments in the first case) for escaping the problem of "guilt by association." Everyone can rationalize that the reason for the problem is the incompetence of the project manager; a new project manager is brought in, a more realistic set of revised project schedules and budgets may or may not be negotiated, and the project continues. Meanwhile, of course, nobody thinks to relax the pressure of overtime work on the technical staff members of the team.

- *Chinese Water Torture*—Rather than facing a high-risk, all-or-nothing showdown near the end of the project, another common game is to bring the bad news to the customer and/or higher management in small pieces. Imagine the scenario, for example, where the project manager's rational estimate for the project is 12 months; with forced overtime and lots of miracles, he thinks it might be possible to finish in six months, but management has imposed a four-month deadline upon the project. Reluctantly, the manager concedes and announces a series of "inch-pebble" deliverables for the project—for example, a new prototype version of the system will be delivered for customer review every week. The first deliverable turns out to be a day late, but the manager reasons that the delay represents 14-20% of the deadline for that deliverable (depending on whether the team is working a five-day week or a seven-day week); thus, he argues that the deadline for the final version of the system should also be pushed back by 14-20%. Management refuses to concede any slippage at this early point, but when the second inch-pebble is also a day late (meaning a cumulative delay of two days over a period of two weeks), the manager repeats his argument. Drip, drip, drip; it's like Chinese water torture—no one single piece of bad news is enough to kill you, but the cumulative effect can be fatal.
- *Smoke and mirrors*—Pity the poor project manager whose higher level IS/IT vice president has hired a metrics consultant with an estimating model that nobody understands. Software metrics are ultimately a form of statistics, and estimating models are based upon sophisticated mathematics. When put in the hands of the innocent, the naive, and/or the politically motivated, these tools can be used to "prove" the validity of almost any estimate. All of this is doubly dangerous if the metrics come from a vendor attempting to prove that the death march project will succeed because of the stupendous productivity of the vendor's CASE tools, visual programming language, or newfangled software-engineering methodology.

- *Hidden variables of maintainability/quality*—This is one of the more insidious games, and it can be played in a constructive or destructive fashion by knowledgeable project managers, higher level IS/IT managers, and/or customers. It's very simple: As a project manager, I can deliver an infinite amount of software to the customer in zero time *as long as it doesn't have to work and it doesn't have to be maintained*. Obviously, it would be foolish to propose a scenario this extreme, but the point is that quality (in the form of defects, portability, maintainability, etc.) is a project "dimension" that has to be taken into account when trade-offs are being considered among time, money, staffing, and other resources. Some customers are too naive to recognize this, and some of them have a very cold-blooded, short-term perspective: "I don't care if the system works two years from now, because I think the business opportunity will be gone—and in any case, I'll be gone. All I care about is that the system has to be available three months from now, and it has to work for 12 months after that." If the political pressure is strong enough, you may find IS/IT managers and the project manager adopting this attitude; it's far less common to see the technical staff members accepting it as a reasonable way of doing business. In the best of cases, this "game" represents the strategy of "good-enough" software that I described in my *Rise and Resurrection of the American Programmer*;[3] in the worst of cases, it's as dishonest and reprehensible as several of the other political games described above.

NEGOTIATING STRATEGIES

What should you do if you find yourself becoming involved in one of the political games described above? Equally important, what should you do if you're an innocent bystander—for example, a technical staff member of the project team—and you observe such games being played all around you as the project deadline, functionality, and budget are being negotiated? Thomsett makes the interesting point that we all learn these political games from our mentors, our managers, and the "elders" of the political culture in our organizations; thus, even if we can't escape the games ourselves, perhaps we can refuse to teach them to our subordinates, in the hope that the whole process of political games will die out after another generation of two.

It's a noble thought, but I'm not so optimistic. I sometimes think that political behavior is genetic, firmly imprinted on our DNA pattern. But even if it's not this bad, the reality is that political games of the nature described in this chapter are all around us; none of this is unique to software projects, and all of us have been exposed to variations on these games throughout our lives. Even if it *was* unique to software projects, there's enough mobility within the software profession that an organization is almost certain to "infected" by highly political managers, vendors, and marketing representatives over a period of time. Political games something are we have to accept as an unavoidable phenomenon and cope with them as best we can.

One thing we *can* do—this also comes from Thomsett's excellent article—is avoid getting sucked into the trap of producing an "instant estimate" for a project. The Spanish Inquisition game is the worst form of this, but there are many lesser forms that take place in the planning and negotiation for death march projects. Whether it's innocent or malicious, the project manager will often be asked for an instantaneous "rough estimate" for the time or staffing required for some aspect of the project; once it's been blurted out in public, it often becomes a hard, unmovable requirement for the project. So, in any situation of this kind, the manager needs to respond with a statement such as "I'll need a day (or a week or a month—or even an hour!) to make some calculations before I can give you an estimate. I'll let you know by email." There are obvious political advantages to being prepared in advance, so that you've already done the necessary calculations before you get hit with the questions; but that's not always possible.

And it's not always possible to avoid the demand for an instant estimate. Suppose you're sitting in a marketing presentation, and the client turns to you and says, "Okay, Harriet, suppose we eliminate the interactive Web browser portion of the system and agree to do the whole thing on our inhouse network, with 10 of our people added to your project team. How long will it take you to get the job done?" All eyes turn to you, and you can see the marketing manager squirming; you probably know from all the discussions that have led up to this question that the politically acceptable answer is, "Three months—no problem!" Chances are that you cannot say, "Gee, I don't really know; we'll have to go back to the office and run that through our estimating model. And I'd also have to interview your ten people to see what their skills are..."

In a situation like this—and even in many of the situations where you *do* have some time to put together a formal estimate—it's crucial to state your estimates in terms of "confidence levels," or a "plus-or-minus" range. If you have absolutely no data with which to construct a detailed estimate, and if the death march project involves completely new technology and unknown people, then it might be prudent to say, "The project will probably take between three and six months," or "I think we can finish in six months, plus-or-minus 50%."

Of course, most project managers are aware of this technique, and they may or may not be using it already. Deciding how large or small the "plus-or-minus" range should be is part of the science of estimating, and I'll leave that to the textbooks listed at the end of this chapter. For death march projects, it's important to keep in mind the *politics* of stating confidence levels during the negotiating process. The most basic political reality, for example, is that anything you say about a plus-or-minus range will be ignored by everyone else that you're negotiating with.

Thus, if you're sitting in a planning session and you tell the customer and various other senior managers, "We should be able to get this project done in six months, plus or minus 25%," everyone will write down "six months" on their note pad.[4] No matter how many times you say it, they'll ignore it; and when your boss feeds the information back to you, you'll find that your deadline is six months. The only thing you can do is *never* drop the plus-or-minus qualifier in any verbal or written statements, promises, commitments, or estimates that you provide. It won't eliminate the problem, but it will provide a cover-your-ass excuse if the project ends up at the high end of your estimate.

One of the readers of the draft manuscript of this edition of *Death March* made an interesting comment along these lines:

> The problem I have with "estimate" is that I understand the word to mean, "best assessment knowing what is known at this moment and what can reasonably be predicted about the future." From experience, what I have seen happen is that once an estimate is published, it becomes a rock solid commitment representing the maximum amount of resources needed to accomplish the project phase or the entire project.
>
> Analysts, programmers, and other technicians understand that an estimate is an estimate. Maybe the project manager understands that too. However, somewhere in the "higher" levels of management, that understanding is transformed, usually into the word, "commitment." Sometimes, prior to project kick-off, top management understands "estimate" too. However, once the initial dollars are expended, an estimate usually becomes a "you bet your job if you do not keep to your commitment."[5]

Unfortunately, there's an uglier aspect of the political negotiation when you introduce the plus-or-minus qualifier into your estimate: You'll be accused of uncertainty, wishy-washiness, weakness, or even incompetence. This is particularly common in the "Marine Corps" style of death march projects discussed earlier in this book. What senior management really wants is a firm commitment—a *promise* that the project will be finished on a certain deadline, with a budget of a certain number of dollars, and a staff of a certain size. This gives them the enormous luxury of (a) no longer having to worry about the problem for the duration of the project and (b) having a convenient scapegoat to blame if the promise is broken. An estimate that takes the form of "X months plus or minus 50%, for $500,000 plus or minus 100%, and with 10 people, plus or minus 25" eliminates that luxury.

Jim McCarthy, in his excellent book, *Dynamics of Software Development* [5], suggests that the project manager needs to confront this head-on and persuade the customers and/or senior management that they need to share some of the burden of uncertainty that the entire project team will be living with on a day-to-day basis. Thus, the project manager effectively says to the customer or the senior management group, "Look, I *don't know* precisely when this project will finish—but since I'm the project manager, I'm far more likely than anyone else in the organization to figure it out as soon as it *can* be figured out. I promise you that once I know, I'll tell you right away."

Only a manager with a lot of self-confidence, *and* the ability to walk away from the assignment, has the *chutzpah* to say something like this in the politically charged atmosphere of a death march project. The time to say it is at the beginning of the project; after all, if the customer and senior management do not respect your ability as a project manager, and if they don't realize that you *do* have a better chance of knowing when the project will finish than anyone else, then why are they putting you in charge of the project in the first place? Are you being set up as a scapegoat? Are you going to be a "puppet manager," with all the decisions being made by other political manipulators in the organization? If so, now is the time to get out!

Similarly, if you're a lowly programmer on the project team and you see political games like this, it may be a strong indication that your project manager (a) doesn't have the confidence to believe in any estimate that he puts forth, (b) doesn't have the backbone to stand up for himself and for the project team, and/or (c) has gotten himself into a political situation where all the key decisions will be made by people who are not directly involved in the project. Again, this is a strong indication that the project is doomed; and before you get too deeply involved, it might be a better idea to seek greener pastures.

Having said this, I'm nevertheless well aware that it's extremely difficult for the project manager to persuade the various "players" to share the uncertainty of the project schedule, budget, and staffing decisions. A savvy customer will indeed do this; a sophisticated IS/IT organization will recognize all of this as an aspect of risk management, which needs to be carried out in a blameless political environment; and human beings who care about and respect one another will agree that it's unfair to make one member of a group carry the ulcer-generating pressure of a high-risk situation.

WHAT TO DO WHEN NEGOTIATING FAILS

In the discussion above, I suggested that if the project manager can't persuade the customer or senior management to share some of the uncertainty associated with the schedule or budget of a death march project, he should seriously consider resigning from the assignment; the same goes for technical members of the project team. But this is only one aspect of a "failed" negotiating process; what should the manager do, for example, if he is 100% certain that the politically mandated deadline of six months cannot and will not be achieved? What should he do if he is 100% certain that the project must have a minimum of three people, but management will provide only two?

I've mentioned the option of resigning a few times already in this book, and I realize that it's not a practical option for some software professionals; indeed, it's more likely to be a problem for the project managers than the technicians, for the simple reason that project managers tend to be five to ten years older and are thus saddled with the impediments of mortgages, dependent family members, half-vested pension plans, and so on. They also tend to be more insecure about their chances of getting another job quickly, while the younger, unmarried project team members are typically much more confident that they can land another job within 24 hours.

It's important to realize here that I'm not recommending resignation as a form of punishment or revenge. It's simply the rational thing to do when faced with an impossible situation and implacable negotiating adversaries. Life will go on; there will be other projects; and there will be other jobs. As Sue Petersen remarked to me in a recent email message:[6]

> *I've learned something from my kids, and I think it applies to work just as much as it does to home life…I* have *to protect myself, my energy level, my emotional and physical health, my quiet-time, and my work time. If I don't protect myself, I won't have anything left for them anyway.*

But there's another issue associated with quitting that needs to be confronted here: the issue of loyalty and the "social contract" between the employer and employee. Up through the 1980s, many software professionals worked in large organizations whose corporate culture involved an assumption of a "job for life." While it was never as strict or as explicit as in Japanese companies, most of the programmers and software engineers at the major banks, insurance companies, government agencies, and computer companies (such as IBM and DEC) assumed that in the absence of war, famine, or plague, they would continue to rise through the organization until they finally retired at age 65 with a gold watch.

Small companies have never had this kind of culture, and many software professionals *have* worked for small companies, especially as computer technology has become so inexpensive that even a Mom-and-Pop grocery store can afford a PC and a Web server. And those of us who have worked for consulting firms, service bureaus, and various forms of entrepreneurial, high-tech startup companies have always known that there is no such thing as a lifetime social contract.

Software professionals in large companies have begun to learn this, too, because the era of downsizing, outsourcing, and re-engineering has caused major disruptions and unemployment in our field. This has been exacerbated by mergers and acquisitions in the computer field and also in highly competitive industries where information processing is a major part of the workforce. When Chemical Bank and Chase Manhattan Bank merged in the mid-1990s, for example, senior management had to deal with the problem of merging two entirely different hardware environments, systems environments, and IS/IT management hierarchy. And as I mentioned in Chapter 1, it's *exactly* this kind of situation that led to many of the death march projects that took place throughout the 1990s.

The problem in many of these large organizations is that while the employ*er* has definitely changed the social contract, the employ*ee* has not reacted accordingly. Many software engineers who have put in 10 or 20 years of loyal service still assume that (a) the company will take care of them, and (b) they should stand by the company, no matter how unpleasant it might be. And "unpleasant" is the operative term for most death march projects: it's not fun sacrificing all of your spare time, working to the point of exhaustion, and coping with stress and political tension. So why do we do it? Because we've signed on for life, and we feel that ethical people should honor their commitments.

However, if the employer has invalidated the social contract, then all bets are off; it's crucial to re-evaluate the relationship and see whether it's worth continuing at all. I certainly don't advocate unethical, immoral, or even amoral behavior—but I see nothing wrong with limiting my commitment to an employer to a period of a year or two, or for the extent of a single project. An employer who says to the death march project team members, "Get this system finished by December 31st or you're fired," is essentially articulating the same kind of "short-term" social contract.

The threat of being fired—which certainly does occur in death march project negotiations—is only one form of "hard-ball" negotiating; threats of being bypassed for a raise or promotion are also common. But if the social contract has been abandoned, and if you're dealing with a hard-ball negotiator in a death march project, then you have the right to play hard-ball, too. And one of the strongest bargaining chips in a negotiating session is your adversary's[7] recognition that you're ready and willing to walk away from the relationship if the results aren't mutually acceptable.

If senior management threatens to fire you if the death march project fails, or if you don't accept the unrealistic deadline they've imposed upon you (which may be two different ways of saying the same thing), then you should be equally cold-blooded in your demands. You may not get them to budge on the deadline, but you can probably be much more demanding than otherwise possible when it comes to staffing your project; I'll discuss this in more detail in the next chapter. And you can *definitely* be more cold-blooded when it comes to ignoring or breaking the administrative and bureaucratic rules and procedures that would other guarantee failure for the death march project.

A variation on this is the old adage of, "Act first, apologize later." It may be a waste of time to "negotiate" a reprieve from the various bureaucratic restrictions that you've decided will hamstring your project. It's certainly worth attempting to do so, because an edict from a high-level manager will usually give you sufficient authority to circumvent or ignore the minions of administrators, committees, and standards-enforcers who will swarm around the project. But if you get a wishy-washy answer—for example, "Well, we're not sure it's a good idea for your programmers to move off-site and have two PCs in their office; we'll check with the Building Services Committee and see what they think"—then stop wasting your time. Just go ahead and do it!

If you're clever, you can probably find a way to circumvent many of the bureaucratic obstacles in such a way that it will take six months for the bureaucracy to notice and to mount an offensive; by then, your project may have finished (or failed) anyway. And if the bureaucracy does mount an offensive, be prepared to play hard-ball: after all, your project is now well underway, and management probably can't afford the risk that you (and the entire project team) will walk out the door and force the project to be restarted. There are two points to keep in mind if you choose this approach:

- You have to be prepared to have your bluff called. If the Methodology Police visit your project and throw a tantrum because you're not using the company's official methodology, you may well get a furious phone call from your boss's boss's boss. You need to be prepared to say, "Mr./Ms. Big Shot, we've decided not to use the methodology because it will guarantee failure. If you feel strongly about this, my team and I are prepared to resign today—otherwise, I'd appreciate it if you would leave us alone and tell the Methodology Police to leave us alone, too. We have work to do." *This won't work unless the senior manager truly believes that you and your team* will *resign on the spot, if pressed.*
- You have to be prepared to deal with enemies who will hold a grudge even if your project succeeds. In the scenario above, you've challenged the authority of the Big Shot manager; he/she won't forget it. You've embarrassed the Methodology Police and made it more difficult to impose their methodology on other victims; they won't forgive you. Indeed, you may have burned so many bridges that at the end of the project, you (and perhaps the rest of the team, too) will be so unpopular that you'll have to quit.

If resignation and "hard-ball" negotiating are not options in your death march project, then what should you do if the negotiating process yields unsatisfactory results? Very simple: Redefine the nature of the project, as suggested in Figure 2.1 in Chapter 2. In the early stages of negotiation, you may have thought you were beginning a "mission-impossible" project: Given adequate resources and a talented staff, you might have been prepared to accomplish miracles. But if you're given inadequate resources and brain-dead programmers, then miracles are not going to occur.

Indeed, it's more likely that you're being pushed into a kamikaze project or a suicide project; only as a variation of the hard-ball negotiating process described above could we imagine that the outcome would be the "ugly" style of project described in Chapter 2. In any case, the key point here is that the project manager has to believe in the possibility of achieving the project goals (e.g., deadline, required functionality, etc.) and must be able to convince the team members of the viability of those goals without "conning" them. As John Boddie [9] points out in a superb book on managing "crunch-mode" software projects:

> The project leader who cares about his people will not try to sell them a bill of goods about the project. He will be honest about the level of effort it will require and its chances of success. Programmers aren't stupid. The experienced ones will have a keenly developed sense to tell them when they're being "fed a line." Most of them won't be a party to project games because they know they are the ones who will shoulder the burden when the crunch comes.

And if the project manager has determined that the death march project goals are *not* viable, but the project has to continue anyway, then it's crucial that the manager explain to the staff members that they are signing on for a suicide or kamikaze mission. Some will accept the mission anyway, and it's important for the manager to understand what their reasons are;[8] but others will resign.

There's an interesting aspect of ethics here. As noted earlier, I don't advocate unethical or immoral behavior of any kind, but I also believe that the negotiations surrounding a death march project almost always force the project manager to deal with the owner/customer and/or senior management as an adversary. The members of the project team, on the other hand, are like one's family. More than just treating the team members ethically and professionally, the manager should feel the responsibility of "taking care" of the team, to ensure that they don't become innocent victims in the political battles. I'm indebted to John Boddie [9] for tracking down a maxim from Napoleon that expresses this thought more eloquently than I could on my own:

> It follows that any commander in chief who undertakes to carry out a plan which he considers defective is at fault; he must put forth his reasons, insist on the plan being changed, and finally tender his resignation rather than be the instrument of his army's downfall.
>
> —Napoleon, *Military Maxims and Thoughts*

NOTES

1.
```
From: Doug Scott, 100072,1276
To: Ed Yourdon, 71250,2322
Topic: Death March Ch2 Queries
Section: The Cutter Edge [14], Forum: CASE - DCI
Date: Thu, Jul 11, 1996, 4:46:20 PM
Ed,
> I'm going to be suggesting in this next chapter that the project
```

```
>manager be sure to identify .....
>Are there any other significant constituencies that I've missed?
I think simply identifying them is a good first step, and then you
need to understand why they would want the project to succeed. Many
don't care, and could thus get in the way. Opponents would stick
out like a sore thumb.
But my biggest single obstacle in deathwatch projects has been my
own management. I came to the UK in 1972, and moved on to big
projects almost immediately. I don't think I learnt anything about
running projects since that date (I learnt a lot about politics,
but that's something else). You need to understand your own manage-
ment's negotiating stance, and if they love to play roll over, you
have to keep them well away from the project.
>2. How important do you think it is for _all_ of the project team
>members to be aware of the existence of these constituencies and
>whether or not they can be viewed as a "friend" or "foe" of the
>death march project?
This has to be managed. In any project, having an external focus to
push against does help to solidify a team. But you mustn't allow
this to stop them helping you. If you need this, I'd say you need to
keep it to single individuals. Deathmarch projects, because of
their size and importance, will usually attract hostility from sur-
rounding people anyway, so it won't be too difficult to create an
enemy - the trick will be to make sure that your potential helpers
aren't all enemies as well.
> *    mission impossible: if we succeed, we live happily ever after
Done that. I don't think I ever classified it as a deathmarch, in the
way I'd normally think of one. But I did develop an ulcer, so... <g>
> *    kamikaze: the project may succeed, but it will kill all of us
Dunno. The certain death is so demotivating, I'm not sure if people
would continue. They'd probably rationalise it into another type of
project.
> *    ugly: the project manager is prepared to sacrifice any
 >      and all of the team members in order to succeed.
Well, I think this comes with the territory. It's part of being a
death march.
> *    suicide: the project has no chance of success, and we're the
scapegoats
Yes, this seems to be one of the fears with death marches.
I don't think I can go along with your matrix, in this case. True
death marches have some characteristics - there is a (possibly
remote) possibility of success; it's so tightly time-boxed that
success within the timescales is difficult to imagine, and one of
the pastimes is to watch announced deadlines being slipped while
still being aware of the need for further slippage.
Personal satisfaction is never high on a death march, and the
chance of success is low - I guess that's what defines a death
```

march. Most death marches fall into your suicide category, I'm afraid. If you had high personal satisfaction and high anticipation of success (which I reckon are correlated anyway), that's not a deathmarch.
As I say, I believe the true differentiator lies in the timescale, rather than in personal feelings. If the timescale is impossible, then you *know* you're on a deathmarch. The only question then is whether you die expensively or slowly.
> How important do you think it is for the project manager to get a
> really good assessment of each team member's level of commitment?
If anyone asks me that question nowadays, I know to run a mile, because that PM will turn the project into a death march. I've never had trouble getting people committed, once I've set up an environment where that commitment will pay results. But I have seen many environments where overtime is regarded as more important than what you're doing (a friend who's just joined Oracle is replete with that attitude now), and I'm not at all impressed by their output.
> _negotiations_. I'll deal with that in Chapter 3
Let me know. when you need stories here. Many are so unbelievable that it's not worth even telling (such as "I don't mind you refusing changes to the design even if it is a fixed price project - all I have to do is ring your chairman, and he'll always tell you do it.").
Doug (back on OS/2 and GCP)

2. From: John Boddie, 73757,3311
To: Ed Yourdon, 71250,2322
Topic: DM Ch2 done, Ch3 queries
Section: The Cutter Edge [14], Forum: CASE - DCI
Date: Fri, Jul 26, 1996, 8:39:15 PM
Ed,
re: if you know of any good negotiating strategies (other than blackmail and torture, which I can't recommend in a book like this <g>), let me know.
The only leverage that the manager has is to bring the risk of failure out into the open and as publicly as possible start postulating fallback positions.
Some useful questions during negotiations,
"If the system is ready on the fifth of September rather than on the first, will we already have declared bankruptcy September second?"
"Is there an 80/20 rule here? If we deliver the critical 20 percent that gives eighty percent of the value, do we need the twenty percent at initial roll-out?"
"Everybody wants things good, wants them fast, and wants them cheap. Everyone knows that you can actually achieve any two of the three. Which two do you want?"
The principle at work is to make those who are demanding the death

```
march look unreasonable if they are unwilling to consider more than
one possible outcome. Unless there is an acceptance of more than
one way to approach the problem, then there is no negotiation. All
the manager can say is, "We'll give it our best shot, but there are
no guarantees."
JB
```

3. *Rise and Resurrection of the American Programmer*, Edward Yourdon (Prentice Hall, 1996).

4. Actually, the politically astute people will take your worst-case estimate and add another "safety factor" before reporting it to their next higher level superior. Your estimate of six months, plus or minus 25%, thus becomes nine months or a year. Unfortunately, the politically naive, or the politically ambitious, will do just the opposite. Thus, the CEO may end up being told that your project will be done in four months or less.

5. Bob Speth email, July 16, 2003.

6.
```
From: Sue Petersen (WWL), 102354,1624
To: Ed Yourdon, 71250,2322
Topic: DM Ch2 done, Ch3 queries
Section: The Cutter Edge [14], Forum: CASE - DCI
Date: Fri, Jul 26, 1996, 6:55:26 PM
>>Another important question I want to discuss in this chapter:
what should the death march project manager do when, in his/her
sincere opinion, the negotiations have failed? At what point does
the manager resign, throw a tantrum, threaten to become the next
Unabomber, etc.? And when he/she reaches that stage, what responsi-
bility does he/she have to the project team, which may have already
begun working? <<
I've learned something from my kids, and I think it applies to work
just as much as it does to home life... I _have_ to protect myself,
my energy level, my emotional and physical health, my quiet-time,
and my work time. If I don't protect myself, I won't have anything
left for them anyway.
Sue P
```

7. Some readers will probably object to the customer's, or one's senior manager's, being described as an "adversary." But the very nature of a death march project implies that the owner/customer, and the various shareholders and stakeholders, are pushing the project manager into decisions that he would not make on his own.

8. It's possible, for example, that a disgruntled staff member may see the death march project as an excellent way of wreaking revenge upon the organization—and he may join the project team in order to make *certain* that the project fails.

REFERENCES

1. Tarek Abdel-Hamid and Stuart Madnick, *Software Project Dynamics* (Prentice Hall, 1993).

2. Barry Boehm, *Software Engineering Economics*, Prentice Hall, 1981.

3. Barry Boehm, Bradford Clark, Ellis Horowitz, Chris Westland, Ray Madachy, and Richard Selby, "The COCOMO 2.0 Software Cost Estimation Model," *American Programmer*, July 1996.

4. Frederick Brooks, *The Mythical Man-Month* (20th anniversary edition), Addison-Wesley, 1995.

5. Jim McCarthy, *Dynamics of Systems Development* (Microsoft Press, 1995).

6. Robert E. Park, Wolfhart B. Goethert, and J. Todd Webb, *Software Cost and Schedule Estimating: A Process Improvement Initiative*. Technical Report CMU/SEI-94-SR-03, May 1994. Pittsburgh, PA: Software Engineering Institute.

7. Robert E. Park, *Checklists and Criteria for Evaluating the Cost and Schedule Estimating Capabilities of Software Organizations.* Technical Report CMU/SEI-95-SR-005, January 1995. Pittsburgh, PA: Software Engineering Institute.

8. Rob Thomsett, "Double Dummy Spit and Other Estimating Games," *American Programmer*, June 1996.

9. John Boddie, *Crunch Mode* (Prentice Hall/Yourdon Press, 1987).