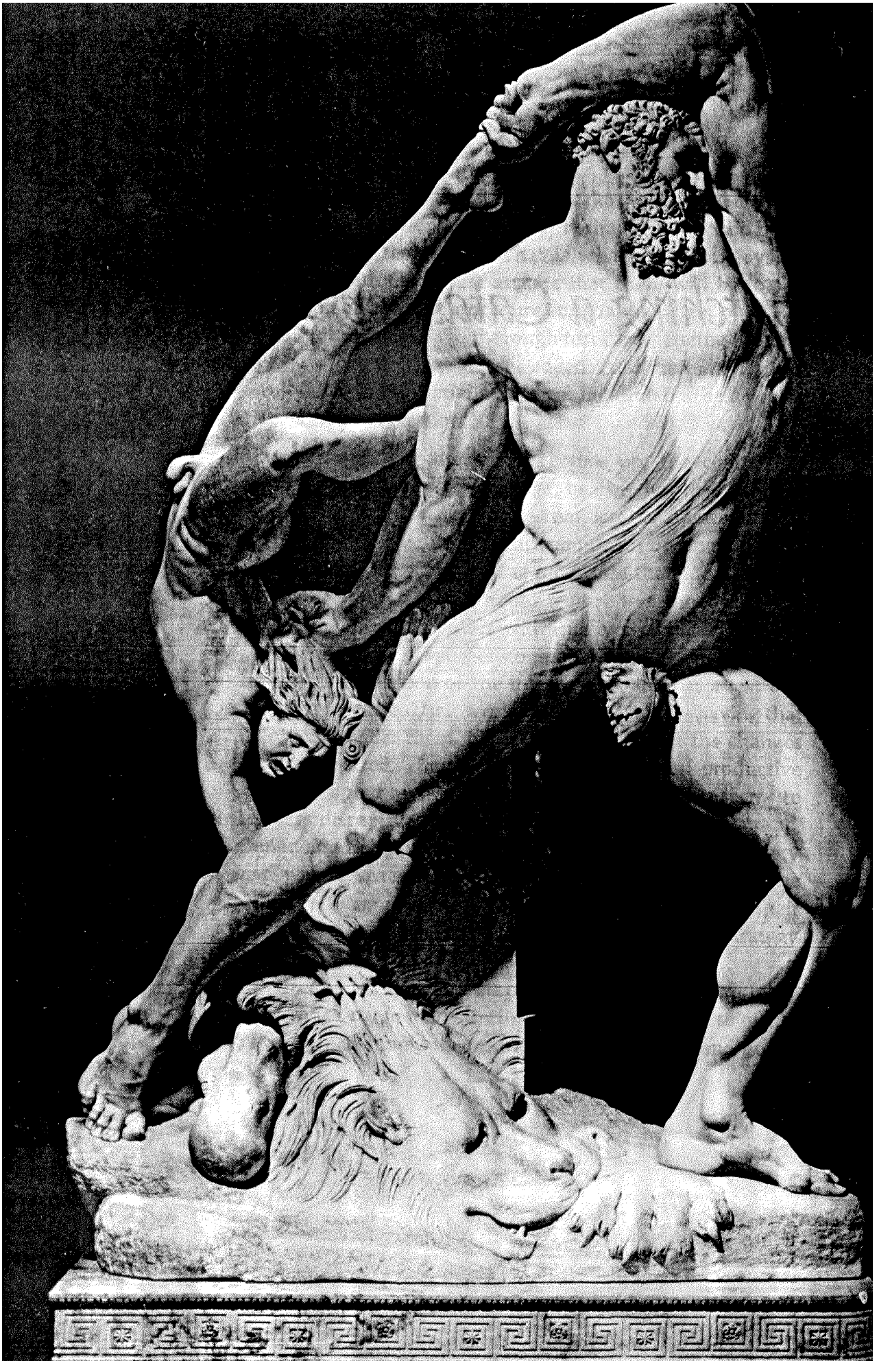
# *14*

# *Hatching a Catastrophe*



# 14
# *Hatching a Catastrophe*

*None love the bearer of bad news.*

***SOPHOCLES***

*How does a project get to be a year late?*
*. . , One day at a time.*

**A. Canova, "Ercole e lica," 1802. Hercules hurls to his death the messenger Lycas, who innocently brought the death-garment.**
Scala/Art Resource, NY

When one hears of disastrous schedule slippage in a project, he imagines that a series of major calamities must have befallen it. Usually, however, the disaster is due to termites, not tornadoes; and the schedule has slipped imperceptibly but inexorably. Indeed, major calamities are easier to handle; one responds with major force, radical reorganization, the invention of new approaches. The whole team rises to the occasion.

But the day-by-day slippage is harder to recognize, harder to prevent, harder to make up. Yesterday a key man was sick, and a meeting couldn't be held. Today the machines are all down, because lightning struck the building's power transformer. Tomorrow the disk routines won't start testing, because the first disk is a week late from the factory. Snow, jury duty, family problems, emergency meetings with customers, executive audits—the list goes on and on. Each one only postpones some activity by a half-day or a day. And the schedule slips, one day at a time.

### Milestones or Millstones?

How does one control a big project on a tight schedule? The first step is to *have* a schedule. Each of a list of events, called milestones, has a date. Picking the dates is an estimating problem, discussed already and crucially dependent on experience.

For picking the milestones there is only one relevant rule. Milestones must be concrete, specific, measurable events, defined with knife-edge sharpness. Coding, for a counterexample, is "90 percent finished" for half of the total coding time. Debugging is "99 percent complete" most of the time. "Planning complete" is an event one can proclaim almost at will.[1]

Concrete milestones, on the other hand, are 100-percent events. "Specifications signed by architects and implementers," "source coding 100 percent complete, keypunched, entered into disk library," "debugged version passes all test cases." These concrete milestones demark the vague phases of planning, coding, debugging.

It is more important that milestones be sharp-edged and unambiguous than that they be easily verifiable by the boss. Rarely will a man lie about milestone progress, *if* the milestone is so sharp that he can't deceive himself. But if the milestone is fuzzy, the boss often understands a different report from that which the man gives. To supplement Sophocles, no one enjoys bearing bad news, either, so it gets softened without any real intent to deceive.

Two interesting studies of estimating behavior by government contractors on large-scale development projects show that:

1. Estimates of the length of an activity, made and revised carefully every two weeks before the activity starts, do not significantly change as the start time draws near, no matter how wrong they ultimately turn out to be.
2. *During* the activity, overestimates of duration come steadily down as the activity proceeds.
3. *Underestimates* do not change significantly during the activity until about three weeks before the scheduled completion.[2]

Sharp milestones are in fact a service to the team, and one they can properly expect from a manager. The fuzzy milestone is the harder burden to live with. It is in fact a millstone that grinds down morale, for it deceives one about lost time until it is irremediable. And chronic schedule slippage is a morale-killer.

## "The Other Piece Is Late, Anyway"

A schedule slips a day; so what? Who gets excited about a one-day slip? We can make it up later. And the other piece into which ours fits is late, anyway.

A baseball manager recognizes a nonphysical talent, *hustle,* as an essential gift of great players and great teams. It is the characteristic of running faster than necessary, moving sooner than necessary, trying harder than necessary. It is essential for great programming teams, too. Hustle provides the cushion, the reserve capacity, that enables a team to cope with routine mishaps, to

anticipate and forfend minor calamities. The calculated response, the measured effort, are the wet blankets that dampen hustle. As we have seen, one *must* get excited about a one-day slip. Such are the elements of catastrophe.

But not all one-day slips are equally disastrous. So some calculation of response is necessary, though hustle be dampened. How does one tell which slips matter? There is no substitute for a PERT chart or a critical-path schedule. Such a network shows who waits for what. It shows who is on the critical path, where any slip moves the end date. It also shows how much an activity can slip before it moves into the critical path.

The PERT technique, strictly speaking, is an elaboration of critical-path scheduling in which one estimates three times for every event, times corresponding to different probabilities of meeting the estimated dates. I do not find this refinement to be worth the extra effort, but for brevity I will call any critical path network a PERT chart.

The preparation of a PERT chart is the most valuable part of its use. Laying out the network, identifying the dependencies, and estimating the legs all force a great deal of very specific planning very early in a project. The first chart is always terrible, and one invents and invents in making the second one.

As the project proceeds, the PERT chart provides the answer to the demoralizing excuse, "The other piece is late anyhow." It shows how hustle is needed to keep one's own part off the critical path, and it suggests ways to make up the lost time in the other part.

## Under the Rug

When a first-line manager sees his small team slipping behind, he is rarely inclined to run to the boss with this woe. The team might be able to make it up, or he should be able to invent or reorganize to solve the problem. Then why worry the boss with it? So far, so

good. Solving such problems is exactly what the first-line manager is there for. And the boss does have enough real worries demanding his action that he doesn't seek others. So all the dirt gets swept under the rug.

But every boss needs two kinds of information, exceptions to plan that require action and a status picture for education.[3] For that purpose he needs to know the status of all his teams. Getting a true picture of that status is hard.

The first-line manager's interests and those of the boss have an inherent conflict here. The first-line manager fears that if he reports his problem, the boss will act on it. Then his action will preempt the manager's function, diminish his authority, foul up his other plans. So as long as the manager thinks he can solve it alone, he doesn't tell the boss.

Two rug-lifting techniques are open to the boss. Both must be used. The first is to reduce the role conflict and inspire sharing of status. The other is to yank the rug back.

**Reducing the role conflict.** The boss must first distinguish between action information and status information. He must discipline himself *not* to act on problems his managers can solve, and *never* to act on problems when he is explicitly reviewing status. I once knew a boss who invariably picked up the phone to give orders before the end of the first paragraph in a status report. That response is guaranteed to squelch full disclosure.

Conversely, when the manager knows his boss will accept status reports without panic or preemption, he comes to give honest appraisals.

This whole process is helped if the boss labels meetings, reviews, conferences, as *status-review* meetings versus *problem-action* meetings, and controls himself accordingly. Obviously one may call a problem-action meeting as a consequence of a status meeting, if he believes a problem is out of hand. But at least everybody knows what the score is, and the boss thinks twice before grabbing the ball.

**Yanking the rug off.** Nevertheless, it is necessary to have review techniques by which the true status is made known, whether cooperatively or not. The PERT chart with its frequent sharp milestones is the basis for such review. On a large project one may want to review some part of it each week, making the rounds once a month or so.

A report showing milestones and actual completions is the key document. Figure 14.1 shows an excerpt from such a report. This report shows some troubles. Specifications approval is overdue on several components. Manual (SLR) approval is overdue on another, and one is late getting out of the first state (Alpha) of the independently conducted product test. So such a report serves as an agenda for the meeting of 1 February. Everyone knows the questions, and the component manager should be prepared to explain why it's late, when it will be finished, what steps he's taking, and what help, if any, he needs from the boss or collateral groups.

V. Vyssotsky of Bell Telephone Laboratories adds the following observation:

> *I have found it handy to carry both "scheduled" and "estimated" dates in the milestone report. The scheduled dates are the property of the project manager and represent a consistent work plan for the project as a whole, and one which is* a priori *a reasonable plan. The estimated dates are the property of the lowest level manager who has cognizance over the piece of work in question, and represents his best judgment as to when it will actually happen, given the resources he has available and when he received (or has commitments for delivery of) his prerequisite inputs. The project manager has to keep his fingers off the estimated dates, and put the emphasis on getting accurate, unbiased estimates rather than palatable optimistic estimates or self-protective conservative ones. Once this is clearly established in everyone's mind, the project manager can see quite a ways into the future where he is going to be in trouble if he doesn't do something.* *

SYSTEM/360 SUMMARY STATUS REPORT
OS/360 LANGUAGE PROCESSORS + SERVICE PROGRAMS
AS OF FEBRUARY 01,1965

A=APPROVAL
C=COMPLETED

#=REVISED PLANNED DATE
NE=NOT ESTABLISHED

| PROJECT | LOCATION | COMMITMNT ANNOUNCE RELEASE | OBJECTIVE AVAILABLE APPROVED | SPECS AVAILABLE APPROVED | SRL AVAILABLE APPROVED | ALPHA TEST ENTRY EXIT | COMP TEST START COMPLETE | SYS TEST START COMPLETE | BULLETIN AVAILABLE APPROVED | BETA TEST ENTRY EXIT |
|---|---|---|---|---|---|---|---|---|---|---|
| OPERATING SYSTEM | | | | | | | | | | |
| 12K DESIGN LEVEL (E) | | | | | | | | | | |
| ASSEMBLY | SAN JOSE | 04/--/4 C<br>12/31/5 | 10/28/4 C | 10/13/4 C<br>01/11/5 | 11/13/4 C<br>11/18/4 A | 01/15/5 C<br>02/22/5 | | | | 09/01/5<br>11/30/5 |
| FORTRAN | POK | 04/--/4 C<br>12/31/5 | 10/28/4 C | 10/21/4 C<br>01/22/5 | 12/17/4 C<br>12/19/4 A | 01/15/5 C<br>02/22/5 | | | | 09/01/5<br>11/30/5 |
| COBOL | ENDICOTT | 04/--/4 C<br>12/31/5 | 10/28/4 C | 10/15/4 C<br>01/20/5 A | 11/17/4 C<br>12/08/4 A | 01/15/5 C<br>02/22/5 | | | | 09/01/5<br>11/30/5 |
| RPG | SAN JOSE | 04/--/4 C<br>12/31/5 | 10/28/4 C | 09/30/4 C<br>01/05/5 A | 12/02/4 C<br>01/18/5 A | 01/15/5 C<br>02/22/5 | | | | 09/01/5<br>11/30/5 |
| UTILITIES | TIME/LIFE | 04/--/4 C<br>12/31/5 | 06/24/4 C | | 11/20/4 C<br>11/30/4 A | | | | | 09/01/5<br>11/30/5 |
| SORT 1 | POK | 04/--/4 C<br>12/31/5 | 10/28/4 C | 10/19/4 C<br>01/11/5 | 11/12/4 C<br>11/30/4 A | 01/15/5 C<br>03/22/5 | | | | 09/01/5<br>11/30/5 |
| SORT 2 | POK | 04/--/4 C<br>06/30/6 | 10/28/4 C | 10/19/4 C<br>01/11/5 | 11/12/4 C<br>11/30/4 A | 01/15/5 C<br>03/22/5 | | | | 03/01/6<br>05/30/6 |
| 44K DESIGN LEVEL (F) | | | | | | | | | | |
| ASSEMBLY | SAN JOSE | 04/--/4 C<br>12/31/5 | 10/28/4 C | 10/13/4 C<br>01/11/5 | 11/13/4 C<br>11/18/4 A | 02/15/5<br>03/22/5 | | | | 09/01/5<br>11/30/5 |
| COBOL | TIME/LIFE | 04/--/4 C<br>06/30/6 | 10/28/4 C | 10/15/4 C<br>01/20/5 A | 11/17/4 C<br>12/08/4 A | 02/15/5<br>03/22/5 | | | | 03/01/6<br>05/30/6 |
| NPL | HURSLEY | 04/--/4 C<br>03/31/6 | 10/28/4 C | | | | | | | |
| 2250 | KINGSTON | 03/30/4 C<br>03/31/6 | 11/05/4 C | 12/08/4 C<br>01/04/5 | 01/12/5 C<br>01/29/5 | 01/04/5 C<br>01/29/5 | | | | 01/03/6<br>NE |
| 2280 | KINGSTON | 06/30/4 C<br>09/30/6 | 11/05/4 C | | | 04/01/5<br>04/30/5 | | | | 01/28/6<br>NE |
| 200K DESIGN LEVEL (H) | | | | | | | | | | |
| ASSEMBLY | TIME/LIFE | | 10/28/4 C | | | | | | | |
| FORTRAN | POK | 04/--/4 C<br>06/30/6 | 10/28/4 C | 10/15/4 C<br>01/11/5 | 11/11/4 C<br>12/10/4 A | 02/15/5<br>03/22/5 | | | | 03/01/6<br>05/30/6 |
| NPL | HURSLEY | 04/--/4 C<br>03/31/7 | 10/28/4 C | | | 07/--/5 | | | | 01/--/7 |
| NPL H | POK | 04/--/4 C | 03/30/4 C | | | 02/01/5<br>04/01/5 | | | | 10/15/5<br>12/15/5 |

Figure 14.1

The preparation of the PERT chart is a function of the boss and the managers reporting to him. Its updating, revision, and reporting requires the attention of a small (one to three man) staff group which serves as an extension of the boss. Such a *Plans and Controls* team is invaluable for a large project. It has no authority except to ask all the line managers when they will have set or changed milestones, and whether milestones have been met. Since the Plans and Controls group handles all the paperwork, the burden on the line managers is reduced to the essentials—making the decisions.

We had a skilled, enthusiastic, and diplomatic Plans and Controls group, run by A. M. Pietrasanta, who devoted considerable inventive talent to devising effective but unobtrusive control methods. As a result, I found his group to be widely respected and more than tolerated. For a group whose role is inherently that of an irritant, this is quite an accomplishment.

The investment of a modest amount of skilled effort in a Plans and Controls function is very rewarding. It makes far more difference in project accomplishment than if these people worked directly on building the product programs. For the Plans and Controls group is the watchdog who renders the imperceptible delays visible and who points up the critical elements. It is the early warning system against losing a year, one day at a time.