Author Contacts: james@satisfice.com, dws@best.com

Plans, Lies, and Videotape

James Bach and Dave W. Smith, SmartPatents

Washington, D.C., seems like an odd place (to the two of us Californians, anyway). Practically every building downtown looks like an ancient monument. From the sidewalk outside the US Supreme Court, the city seems like it was built not by humans but by some alien race. The tourists, workers, and even dignitaries who scuttle up and down the granite steps just don't fit the scenery.

We feel the same way about most of the software project plans, schedules, and process descriptions that we read. They seem more like image advertising than technical information. They don't fit the behavior of the people doing the project and they don't describe reality. And it isn't just the documents. Consider this dialogue James recently had with a project lead:

James: "When will you be feature-complete?"

Project lead: "We're shooting for Friday."

James: "Well, I know that's been the plan, but I just read in your e-mail that the project is a week behind schedule."

Project lead: "That's true. Yes."

James: "So when you say that you're 'shooting for Friday,' you actually mean that, although you have no confidence that you will deliver on Friday, you hope that the test team will forgive you and appreciate how much you wanted to deliver it then. You also hope that we appreciate how hard you are working to keep the schedule from slipping out any farther than the Friday after next. Does that sound right?"

Project lead: "Uh, yes."

Why are smart, skilled, responsible, experienced software people like us so often mixed up about commitments?

This statement—shooting for Friday—sounded like a statement of fact or a confirmation of commitment, but turned out to be something almost the opposite. We can think of many similar examples: a list of "should have" features turns out to be "won't haves"; a test plan states "test all possible combinations of X and Y ..." even though the tester knows that there will not be enough time to examine more than a tiny fraction of those combinations; another tester writes 400 test cases, puts each one under source control, then only executes a hundred of them because "the rest aren't worth running"; a project plan states that it is critical to have measurable goals for the project, then offers a set of vague goals with no mention of how to measure them.

Why don't we just speak plainly, then do what we say we'll do? Why can't we just follow our plans? These are old questions, of course, but we're tired of the pat answers that usually follow.

Sure, sure, some managers are shallow and uneducated. Some developers are untamed cowboy technocrats. Some people don't care. Enough already. Since little of that applies to the kind of people who make the effort to read *Computer*, we're really interested in a different question: Why are smart, skilled, responsible, experienced software people like us so often mixed up about commitments?

Jerry Weinberg devotes much of his book, *Quality Software Management, Vol. 3, Congruent Action* (Dorset House, 1994), and his workshops (http://www.geraldmweinberg.com) to examining this question. Other authors seem content to ignore the issue. Let's do our part to reverse the trend.

A PUZZLING EXAMPLE

Let's dissect a real example of a conversation about commitment, caught on video. At Weinberg's Software Engineering Management workshop, we took part in a brief project simulation that involved a new employee, Bill, joining a team. It starts with the project manager leading Bill to a table. On the table are several jigsaw puzzles. Three other people sit at the table piecing them together. This dialogue is transcribed, verbatim, from the tape:

Manager: "As we agreed in your interview, you're going to be my border man. You're going to put the borders together." [gestures at the puzzle table]

Bill: "Okay, what kind of schedule are we under for borders?"

Manager: "I expect you to do it as fast as you possibly can. I'll come back and check on your progress." [turns to walk away]

Bill: "Okay. But what's your expectation of finish time?"

Manager: "I expect you to be done in about ten minutes."

Bill: "Okay, let me get started on this and see if this is like other borders I've done. Borders do vary across—"

Manager: "Sure they do."

Bill: "We'll see if we need to adjust—"

Manager: "Okay, great." [turns and leaves]

Bill sits down at the table in front of a small pile of puzzle pieces. Twelve minutes later the simulation ends. Though the three other folks at the table had each put parts of the puzzles together, Bill had not. In fact, throughout the exercise, he hadn't once taken a close look at any of the pieces on the table. Instead, Bill spent his time talking to the other puzzle solvers and attempting to talk to the project manager.

> Eliminating ambiguity through the use of rigorous specifications and methods is a tactic directly at odds with the short cycle times required for modern, market-driven software development.

Certainly, from the point of view of the project manager, Bill did not even begin the task he agreed to do. The fellow playing the manager had little trouble feigning indignation at his new employee's lack of visible progress.

PIECING IT TOGETHER

During the simulation debriefing, observer comments about Bill's behavior ranged from "he was avoiding the thing he was told to do" to "he looked bewildered" to "I was surprised by the line of logic he was pursuing." Most of us expected Bill to put puzzle pieces together. We were a bit surprised when Bill revealed that he had committed, in his own mind, to a larger goal: helping the team solve the puzzles.

This larger goal, although not stated explicitly in the simulation, struck Bill as more professional and responsible than his explicit and narrowly focused assignment. This innocent difference in the focus of the commitment caused him to behave in ways that were unexpected in the eyes of those who focused only on puzzle pieces.

In light of Bill's actual commitment, his actions during the simulation—seeking team members' role definitions and looking for how his role would interact with them—are understandable. Bill quickly perceived a problem in the process being used to solve the puzzles. He noticed that the other team members had apparently laid claim to sections of the puzzle, including borders. Bill attempted to negotiate with them to get those pieces, but was rebuffed. Then he attempted to escalate his concerns to his manager.

The manager wanted none of it, saying "I'm not really ready to talk about that yet. I expect you to work it out with the team," and seemed concerned instead about a small pile of disconnected border pieces in the middle of the table. From Bill's point of view, the manager reneged on an agreement (implied by "Okay, great.") to allow the task and schedule to be revisited. Thus, the behaviors of both actors in this drama were completely consistent, given their individual views of reality. And yet there was conflict.

You might be tempted to dismiss Bill's redefinition of the task as an irresponsible act. We don't think so. It may be reckless to form private interpretations of tasks in a well-defined and supervised environment like a fast-food restaurant or an assembly line, but software projects are almost never that definitive (or that closely supervised). Individuals are required to use their judgment and initiative (or heroism, as James likes to call it).

It's also important to realize that this particular simulation, unbeknownst to Bill, was designed as a trap. Had Bill done exactly as he was told, he would have been thwarted by the very process problems that he identified. In one sense, he beat the simulation by delaying the assigned task while he sought to understand and correct the team's process.

WHY COMMITMENTS GO WRONG

On the plane ride back from the seminar, we were inspired to make a list of reasons for why otherwise capable, well-intentioned people appear not to meet their commitments. Many of the reasons we listed relate directly to the puzzle project simulation.

No commitment in the first place

What signals tell you someone has agreed to do something? In most cases, they are pretty ambiguous: a nod, a word, or the absence of an objection. Probably not a written contract, and even then, legal phrasing is itself subject to interpretation. That's why we must periodically refresh and reconfirm important commitments.

Conditional commitments, with conditions unmet

Look closely at Bill's words. He never actually said that he would put any pieces together. What he indicated agreement about, at least implicitly, was to analyze the border problem and perhaps to suggest adjustments. During the debriefing, he clarified "adjustments" to mean "adjustments to the schedule." You can spot the condition hiding in Bill's agreement. He implies that if this puzzle border problem isn't like others he's done, then he may not be able to fulfill the task within the allotted time.

Commitment to work toward a goal, rather than achieve it

Conditions are so changeable and goals are so volatile in software development that it becomes a survival skill to commit oneself by degrees, rather than all at once. Total and instant commitment saps a lot of energy, and wreaks havoc with prior commitments. One way to do this is to commit to a level of effort, rather than a result.

The "shooting for Friday" line seems like an example of this, since the developers were working toward the goal even though they knew they would not achieve it. Also, Bill's repeated use of the ambiguous word "okay" turned out to be an indicator of a commitment to work toward the puzzle solutions, rather than specifically to put pieces together. This kind of ambiguity leads to very unpleasant surprises during the release cycle.

Different impressions of the true commitment

Bill, his manager, and many observers had a different impression of the commitment. It helped to have the videotape to review the exact words, but even then the actual words were open to diverse interpretations.

Commitments made under duress

Often we find ourselves in situations where we cannot yet commit, but overtly

Continued from page 106

refusing to commit causes a lot of commotion. It takes substantial courage and energy to raise your hand and stop everything. Bill may have felt that he didn't have the option of saying "No. I can't agree to this assignment until we figure out how to work together."

Commitments made in error

We may commit without enough information about the implications of that commitment. A common manifestation of this is committing to a task without adequate consideration of the total load of all the other tasks to which we've already committed. We may be able to do anything—that is, any one thing—but we can't do everything at once.

Commitments not worth the effort to fulfill

Sometimes it's better to quit. Unfortunately, there may be no protocol for withdrawing from a commitment, and therefore no graceful way to do it.

Terms of commitment changed midstream

We expect technical people to use their common sense and their best professional judgment. Sometimes this expectation results in the appearance of shifting commitment as people reinterpret their commitments in light of past experience, risks, related tasks, and new information.

Reinterpretation is a good thing! We should be glad that humans, unlike computers, have DWIM (do what I mean) capability. Still, DWIM is a difficult art, requiring continuous adjustment as conditions change. To succeed, a high-performance team needs a shared protocol for recommitment. In the simulation, Bill and his manager did not develop such a protocol.

In his article "Unresolved Ambiguity, The Silent Source of Risk in Your Project" (*American Programmer*, Apr. 1996), Brian Lawrence argues that ambiguity is the root of most important problems in software projects. He suggests some straightforward approaches to resolving ambiguity. We agree, but reducing ambiguity can take a lot of time, skill, and effort.

Eliminating ambiguity through the use of rigorous specifications and methods is a tactic directly at odds with the short cycle times required for modern, market-driven software development. That sort of rigor also requires a level of excellence in teamwork and technical mastery that few development groups attain, although many shoot for it.

> The puzzle project simulation we participated in was designed to be a no-win situation. Poor Bill never had a chance.

The puzzle project simulation we participated in was designed to be a no-win situation. Poor Bill never had a chance. Many real-life projects are similarly fraught with conflict. But the only real failure in these situations is the failure to learn. Bill and his manager have work to do to clarify their commitment protocol. We are striving to do the same at SmartPatents. Meanwhile, our VP of Engineering, Luke Hohmann, reminds us that intentional commitment to a seemingly impossible goal (like putting a man on the moon) is sometimes a way to achieve dramatic breakthroughs in performance.

Looks like we have our work cut out for us. But that's a subject for another article. We promise. Really. ❖

Acknowledgments

We thank Brian Lawrence, Johanna Rothman, Luke Hohmann, and the facilitators and participants of Jerry Weinberg's Software Engineering Management '98 group.

Dave W. Smith is a senior developer at SmartPatents. Smith has led projects at a number of Silicon Valley startups and has participated on both sides of many ambiguous commitments. Contact him at dws@best.com; http://www.best.com/~dws.

ture—including the database schema, mapping to the object layer, application functions, client-server communication functions, and Web-based user interface.

Data migration

Migrating legacy data is one of the most difficult tasks in replacing a legacy system. This difficulty stems from the nature of the legacy data and the highly constrained RDBMS design. The Domain IDL and metadata approach made it easy to tackle the most difficult issues: data validation and correction. To accomplish these tasks, we used the Domain Object Definition Language (ODL).

As shown in Figure 3, our process filters legacy data and reformats it into ODL files that conform to the metadata extracted from IDL. Next, Informix High-Performance Loader (HPL) files were generated from the ODL files and loaded into the database. With this approach, we successfully migrated 10 million objects (75 million database records) in three months, during live operation. This minimized the manual effort in validating and cleaning the legacy data.

Domain successfully replaced the largest GTE legacy system, one that has been in operation for decades, in less than 18 months. This accomplishment required more than just state-of-the-art technology: It required a synergistic approach that brought together hardware, software, and people. This approach combined advanced software techniques with practical solutions to achieve a common goal shared by users, developers, and managers.

Now, with a state-of-the-art infrastructure, GTE can easily and quickly expand Domain to meet shifting operational needs and the new business challenges facing telephone service providers today. ❖

Scott Bollig is the manager of the International and Complex Systems Department at GTE Laboratories in Waltham, Mass.; contact him at sbollig@gte.com.

Dan Xiao is the Domain program manager at GTE Laboratories; contact him at ds24@gte.com.