Presentation Notes
Paper
Bio
Return to Main Menu

PRESENTATION

W1

Wednesday, November 3, 1999
1:00 PM

You Want It When? Negotiating Test Schedules

Greg Pope
Azor, Inc.

International Conference On
Software Testing, Analysis & Review
November 1-5, 1999
San Jose, CA

You Want it When?

Estimating and Negotiating Test Schedules

By:

Greg Pope, AZOR, Inc.

1032 Elwell Ct. Suite 240

Palo Alto, CA 94043

650-934-2869 / greg@azor.com

Given at:

STAR West, San Jose, CA

November 3, 1999

Heard This Before ?

- It is January 1st and marketing has an idea
- “Must get the new product out by June 1st”
- “Development says it will take 5 months”
- “That leaves a month to test, you can do it”
- “We will use the FAD methodology”

FAD Methodology
(Fantasy Application Development)

- Step 1 - Announce product and release date
- Step 2 - Design logo and make T-Shirts
- Step 3 - Determine what the product is
- Step 4 - Estimate development time
- Step 5 - Write the code and web pages
- Step 6 - Write the spec (optional)
- Step 7 - Beta release (ready or not)
- Step 8 - Give incomplete version to Test
- Step 9 - Announce upgrade program and patches

What If Other Industries Used FAD ?

- **Pharmaceuticals - Here is a new drug, works great on frogs and pigs, you can try it at no charge. (Beta Program)**
- **Automotive - It is a great car, and in 3 months we are updating it (a small fee for you) to add the trunk and reverse gear. (Upgrades)**
- **Aviation - Welcome aboard, today we are going to be the first aircraft to try the new GE engines with the old Bendix flight controls. (Platforms)**
- **Airlines - Sorry for the delay, but the latest version of our cockpit software no longer supports landing at our original destination, so we are going to Pittsburgh instead. (Version Compatibility)**

Informal Survey - Test Schedule (1993-1999 - 6,000 Developers/Testers)

- Testers having too much time to test - 0
- Testers having the right amount of time to test - 250
- Testers having too little time to test - 5,750

Why Incorrect Estimates Lengthen Project Time

- Time's up before requirements are understood
- Times up before design is complete
- Coding begins with fuzzy requirements, incomplete design
- Developers may make requirement and design decisions in their own best interest
- Unit test may become debugging
- Test schedule shortened to hold end date
- Product-complete milestone slips to delivery date
- Acceptance of the overrun

For More Information: "Software Engineering Economics", Barry Boehm

Pope's Estimating Laws

- Law 1 - Software takes as long to develop as it takes
- Law 2 - Software development time is not influenced by what we would like it to take
- Law 3 - Estimating a shorter time than it will actually take only defines the length of the overrun

Countermeasure - Negotiating Skills

- Requires acceptance of problem
- Not taught in colleges
- Only having one negotiating style is limiting ("You will ship it over my dead body.")
- We negotiate all the time anyway other places
- Leads to more Win-Win solutions

Negotiation - Definitions

- **Exploration to formulate viewpoints.**
- **Delineate areas of agreement or contention.**
- **Working out practical arrangements.**

For More Information:
Gerald I. Nierenberg- ***Fundamentals of Negotiating***
Nightingale-Conant - ***The Art of Negotiating***

Negotiation Examples

- Negotiating a test and development schedule with the program manager
- Ordering a meal at a restaurant
- Setting up a meeting time
- Obtaining the release of hostages

Negotiate What?

Product	Testing Service
Price	Schedule
Delivery	Budget
Features	Staff
Shipping	Tools
Terms	Release Dates
Follow-on	Acceptance
Warranty	Environment

Success Likelihood

- The issue is negotiable (buying a car is, selling your child is not)
- The negotiator's interest in giving and taking value, and compromise (toll taker)
- Negotiating parties trust each other to some extent
- Trust - Words and Actions match

Negotiation Essentials

- Knowledge of human behavior
- Preparation, know all the related facts on both sides of the issue
- Understanding of techniques, strategies, and tactics
- Understanding the needs of both sides, both direct and indirect.

Example Preparation

- **Collect previous actuals on:**
  - **Defects per KSLOC**
  - **Hours to find defects**
  - **Time to run tests**
  - **Pass and fail percentage**
  - **Defects found before and after delivery**
  - **Classification of defects (requirement, coding, system, testing, documentation)**
  - **Accuracy of prior estimates**
  - **Customer satisfaction survey results**
- **Know style of the person negotiating with**

Directness and Openness

- **Direct - Do it and do it now per these instructions.....**
- **Indirect - You should have known what I was thinking.....**
- **Open - I feel angry and disappointed that we have not been given time to test the new build.....**
- **Self Contained - Everything is fine.**

For More Information "Relationship Strategies" by Alexandra and Cathcart

Relationship Strategies

	Indirect	Direct
Open	Relater	Socializer
Self Contained	Thinker	Director

Director Style

- Self-contained, direct
- Most Important - The bottom line on the issue, strategies
- Wants things to be measured, likes competition, work first
- Assertive, Responsible, Straightforward, Practical, Self Motivated
- Needs to be in charge of others, imposes their standards on others
- Usually does not have a "product," juggles many things at once
- Usually migrates to manager positions
- Black and White approach, cutting edge,
- Jack Lord in Hawaii Five 0, Barbara Walters, Managers

Thinker Style

- Indirect, self-contained
- Most Important - The logic behind the issue, the details, the rules
- Detail oriented, methodical, predictable, dependable, precise
- Likes to engage in intellectual debates, organized, loyal, orderly
- Usually introverted, has "show me" attitude, focused
- Likes to work alone, has the right tools, loves to gather data
- Perfectionist, less interested in outcome, problem solver
- Jack Webb in Dragnet, Joyce Brothers, Data on Star Trek, Developers, Testers, Accountants

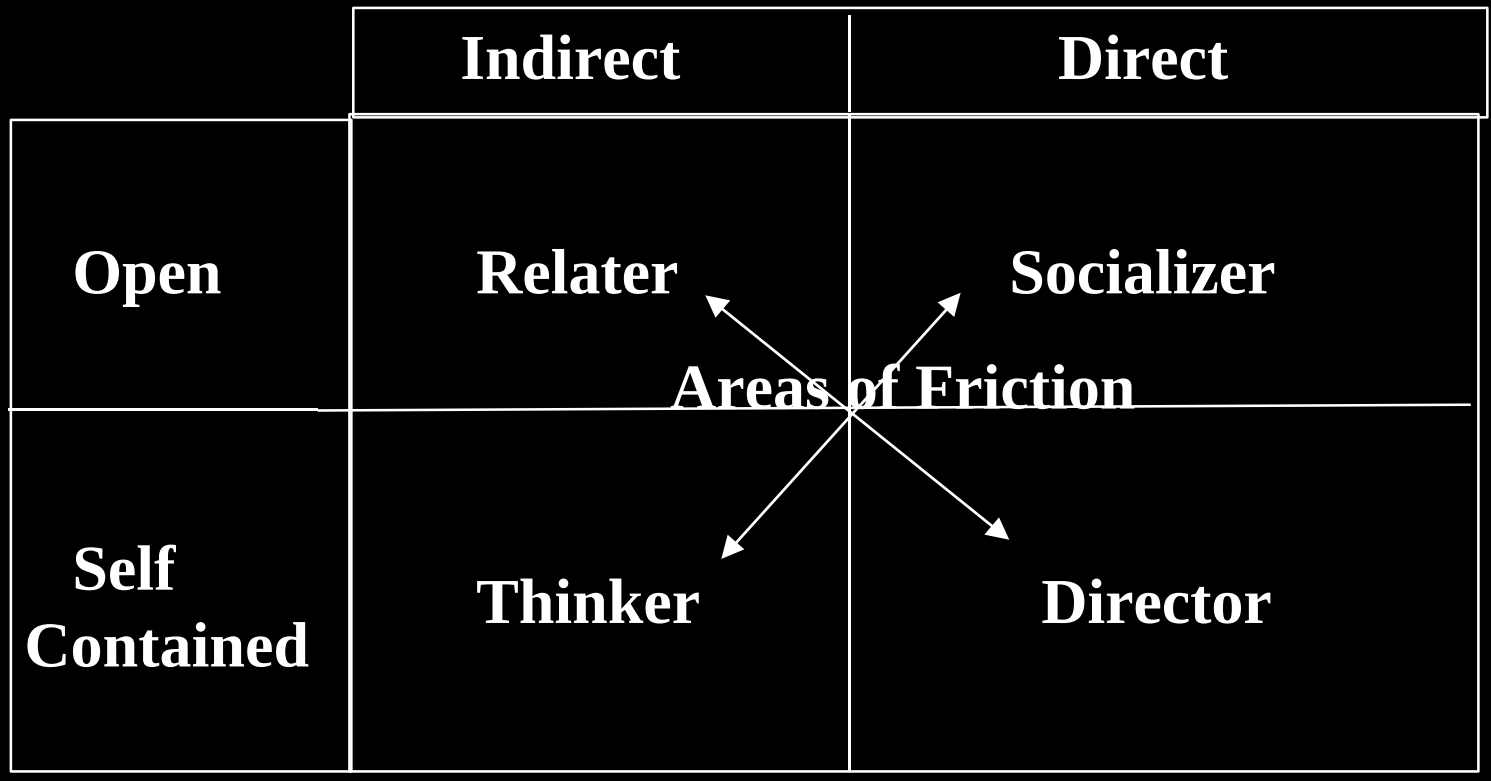
Socializer Style

- Most Important - Interactions between people
- Direct - Open
- Flexibility, goes with the flow, avoids formal plans
- Goes for the gusto, jumps in and takes charge, ready, fire, aim
- Avoids bureaucracy, likes challenges, very optimistic
- Hates to work alone, likes to be where the action is
- Likes recognition, Egotistical, Impatient, Impulsive
- Mickey Rooney as Andy Hardy, Dom Delouise, Sales and Marketing persons

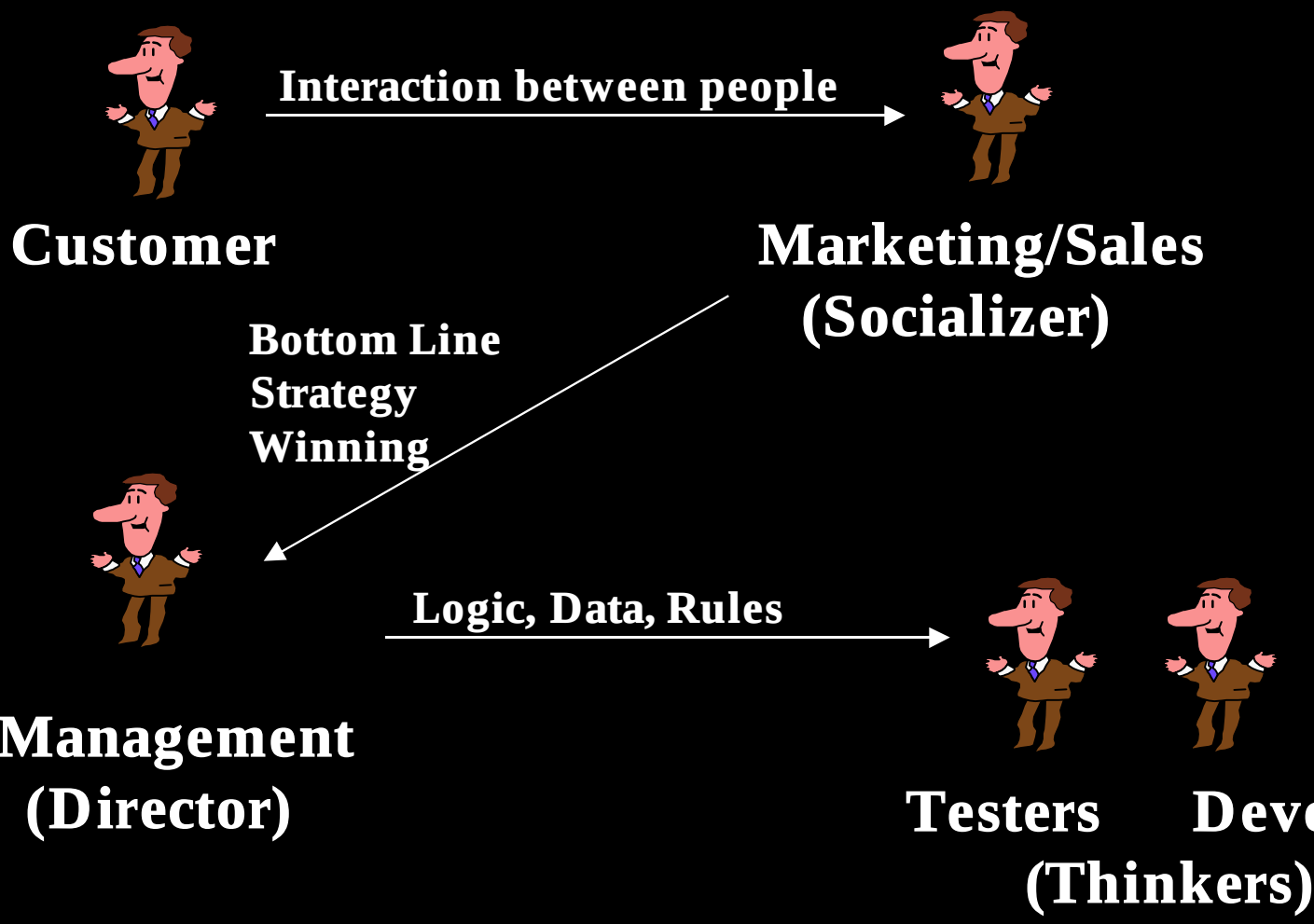
Relater Style

- Most Important - Knowing how everyone feels about the issue
- Open, indirect
- Good listener, empathetic, emotional, helpful
- Flexible, goes with the flow, team player
- Not the job that counts, it is the people
- Likes the personal approach, not formal
- Warm, reliable, patient, relaxed
- Mr. Rogers, Marcus Welby, Mary Tyler Moore, Therapists, Nurses, Physicians

Dealing With Humans



Software Development Communicating and Styles



Dealing With Style

- Be aware of your own style
- Be aware of the style of the people you must interact with
- Be aware of the company's style
- Present information in the same style of the person you are negotiating with to reduce friction

Selling Test Plan to Marketing

- The Test Plan is flexible
- Gives awards and recognizes achievements
- Eliminates bureaucracy and paperwork
- Allows frequent participation in meetings
- Participation in major milestones
- Options for using contractors, outsourcing, customer service as testing resources
- Show how test plan focuses testing on frequent 800 line complaints

Selling Test Plan to Developers

- Test Plan is logical, rule based
- Test Plan details have been thought out
- Test Plan contains checklists and templates that simplify work
- Test Plan has been demonstrated to work on a pilot program
- Test Plan provides the right tools for the job
- Test Plan provides data to track progress

Selling Test Plan to Management

- Test Plan follows a strategy (use case based, risk based, top down, etc.)
- An attractive return on investment to the bottom line ($4,000 per defect)
- Shortest possible test time with reasonable risk
- It is a better Test Plan than the competition's
- Assumptions and dependencies clearly stated
- Measurable and highly visible milestones
- Early detection and correction of estimates

Favorite Tactics For Negotiating Test Schedule

- Columbo - play dumb, ask questions, reflect back
- Trump - bracketing estimates
- Reversal - opposite of what is expected
- Horsetrading - trade time for tools, outsource, temps
- Gasp (and Counter Gasp)
- Switch Hitter - use the best people
- Limited Authority - don't shoot from the hip
- Participation - group decides
- Salami - get what you need *poco a poco*
- Pinocchio - the last resort

Summary

- The biggest obstacle in the software industry is lack of enough time to do our jobs well
- We apply our education, experience, and tools, but the problem persists
- We may not think of negotiating as an option because it is not part of our education
- Negotiating can buy valuable time, help management avoid blunders
- Negotiating can make the job less stressful and more creative
- We will never get things our way completely, and good thing, else nothing would get shipped.

You Want it When? Estimating and Negotiating Test Schedules by Greg Pope

Learning from the wisdom of Owl.

From our beginnings, humanity has been afraid of the night, the dark, the unseen. Software, too, is unseen, and those who establish the schedules for its development and testing are often wandering in the dark. That's why every software developer and tester needs Owl-like Wisdom.

Native Americans called Owl the "Night Hawk." Owl can see things others can't. Owl can hear things others don't hear. Owl's flight is silent; his prey never hears him until he strikes.

The Owl has been associated through the ages with wisdom and clairvoyance, which are the essential underlying principles of this article. For much of my career as a software developer and tester, I could see and hear things others were missing. I'm sure you've had the same experience—especially when the marketing guy begins a conversation with, "All we have to do is …" I would argue with marketing, finance, and management, trying to show them what they couldn't seem to see. But it wasn't until I adopted the wisdom of Owl and began to see what I *was missing that I started to get the time I needed to do the job right.*

With Owl's Wisdom, no one can deceive you no matter how hard they try. With Owl's Wisdom, you have the power to see past ulterior motives and hidden agendas. With Owl's Wisdom, you can pin point your target instead of flailing around in the dark. But Owl Wisdom is powerful and must be used with caution; you may frighten those whose blindness you reflect.

***

Here's a scenario that is probably familiar to every experienced software testing manager on Earth: It starts with an enthusiastic VP of marketing, anxious to stay competitive, promises delivery of a new product by a certain date. The date falls within that critical period called the *marketing window of opportunity*. Failure to deliver within that window will result in a loss of market share—a fact of life in the high-tech fast lane that keeps the whole marketing department awake at night.

Having promised to make the window, the marketing VP then seeks out the head of development and asks how long he or she thinks it'll take to build the promised software. This notoriously optimistic soul recognizes the need to make the window and senses the urgency in the VP's voice, and so says something like, "We can do that in six months." Never mind that it'll really take ten months, the VP has heard what he needed to hear, and all is well.

Feeling warm and secure, the marketing VP now wanders over to the test group. More often than not, he ends up going to lunch instead, but in this scenario he actually stops by and asks how long it's going to take to test this software. Whatever the test manager says—typically not much—the marketing VP doesn't really hear; his mind is floating blissfully over a completely unrealistic release date.

And that's how software companies find themselves sprinting through a development schedule that's too short, sending out software that's too buggy, and spending lots and lots of money to fix things that should have been fixed before the product ever went out the door.

Time: The Most Precious Commodity

Over the years, I've talked with thousands of people engaged in software development, testing, and QA, and I always find a way to ask them about their biggest problems. Is it the testing tools? Do they need better training? Would they be helped by a better understanding of the process? Without exception, the first thing they want to talk to me about is *time*, or rather, the lack of sufficient amounts of it to do their jobs properly.

The initial or analysis phase of software development is, in large measure, about asking and answering questions: What does this particular customer want? What are the new features we're putting into this version? What are these new features supposed to do? How fast do they have to operate? Will the software work in other environments? Are there any industry standards or guidelines that will influence this product? What are the interfaces like? With what other pieces of software or other devices do we have to interface?

In today's often savagely truncated development and testing schedules, you're barely halfway through the process of answering vital questions like these when the time's up. Now you're going to design and code, because after all, writing code looks like real work (to unseasoned management, anyway). With at best a partial understanding of what this thing is supposed to do from the customers' viewpoint, you now have to come up with the architecture, design, and code. You have to allocate which pieces and functions go into which modules or objects. You have to code it yourself or assign those modules to somebody else to code. You have to understand how the database will be designed, understand the protocols to be used, decide what the GUI will look like, decide what can be reused.

Consequently, a group of well-intentioned software engineers, working under the gun, ends up making customer-based requirement decisions and high-level design decisions *in the middle of writing code*. With incomplete requirement analysis and a partial (at best) design, their decisions are necessarily based on whatever is most convenient for them. Chances are, if it's easy to code it in, say, C++, it will get done. If it is hard to code in C++, it won't. Coding software is hard enough when the requirements and design are well understood.

Meanwhile, development is taking longer than expected and that marketing window is looming large on the software development manager's horizon—and so is that marketing VP, whose butt is on the line. Now, the only way to make that window is to cut into the test schedule. I call this The Accordion Effect: as the software development process expands, unseasoned project managers often make up for it by decreasing the time allowed for testing. And that's where we get into the real trouble.

The resulting product is often full of stuff the customer doesn't want. The product isn't robust, won't handle errors very well, and won't interface with other devices very cleanly. And it's buggy. Buggy,

buggy, *buggy*. And that means greater expense down the road, most of which could have been avoided with the right schedule and adequate testing.

Negotiation 101

In this article, my goal is to introduce an idea that might seem a radical concept to many in the development and testing community, namely that the responsibility for securing adequate time for integration and system level testing in the software development schedule rests with *the test engineers and their managers*, and that securing time for adequate unit testing and upstream inspections *rests with the development engineers and their management*. It's not fair and it's not right, but I believe the onus is on those who understand the testing process best.

When I was a first-year programmer, I learned early on that management couldn't *see* software, and how much that simple fact influenced their perceptions of the work of programmers. I was working side-by-side with a fellow whose programs did a lot of IO on the magnetic tape drives. When people walked by the computer when his software was running, they'd see all four tape drives spinning, and they'd say, "Just look at that! Boy, that's really good software." My software, on the other hand, was computationally intensive, prompting comments like, "I don't know, his software doesn't seem to be very efficient." The naked stupidity of the situation infuriated me at first, but I eventually began putting lots of tape rewind commands in my software between calculations, just to keep the tapes spinning. As soon as I did this, I began to get lots of compliments from management.

To unseasoned project managers, writing code looks like real work; analyzing requirements or designing tests doesn't. When the software development team is working against a hopelessly unrealistic schedule, project management tends to emphasize writing code; they may not support you "wasting" time doing things like analyzing requirements and designing tests. On a software project, less experienced project managers often fail to grasp that activity is not necessarily progress.

As software development and testing professionals, we ultimately must educate the other segments of our industry, to train their ears, so to speak, to the relative pitch of our part in the software song. It seems as though we've all gotten used to the sour notes; we need to be listening for the harmony.

To accomplish this task, we need to develop a set of skills usually thought unnecessary in this field. In an nutshell, we must master the art of effective negotiation.

It wasn't until late in my career that I came to understand the importance of developing negotiating skills. In the early days, whenever I was hit with yet another impossible deadline, I simply got angry and jumped in someone's face. Needless to say, my effectiveness was marginal at best, and management always seemed to be mad at me, although I couldn't understand why at the time. Wasn't it the weasels in marketing who were making all the promises we couldn't keep? Hey, buddy, I'm just trying to get enough time to do a good job!

Eventually, I recognized that I was lacking some essential skill set. I mean, when has anyone ever talked to a software or test engineer about developing negotiating skills? Most of the test engineers

I've met or worked with over the years still find this concept utterly foreign. In time, I got some training. I read books, went to seminars, took courses, and talked one-on-one with people I knew personally to be very effective negotiators. It wasn't long before I acquired the skills and strategies I needed to fight back *silently and effectively*. I found myself with more time to do the job right, and my name wasn't on everyone's hate list.

As the saying goes, there's more than one way to skin a cat, but the following negotiating techniques have served me well in the software testing arena. Master these, and you'll be better prepared the next time the marketing guy stops by on his way to lunch.

What is a Negotiation?

Negotiating is something we do almost every day, although most of us don't recognize that we're doing it. When you order a meal in a restaurant, you might ask for the number eight special, only with a chicken burrito instead of the bean burrito. Or maybe you want it served to you on two plates so you can share it with your friend. To get it your way, you negotiate a little when you order. Or maybe you're going out with your spouse tonight and you want to see an action movie, but she wants to see a love story. Which flick do you see? That depends on the outcome of your negotiation.

We're always negotiating. We do it at home, at work, in the car pool lane, by the water cooler. Only once in my career did I ever sit at a table in a conference room and conduct what could be called a formal negotiation. (That was at well-known aircraft company for a $12 million contract.) More often, we negotiate in the hallways, the bathroom, before a meeting, after a meeting, or over the telephone. We catch somebody heading out for lunch and buttonhole him or her about the schedule.

In short, negotiations are going on all the time. Good negotiators recognize this, and stand ready to apply their skills whenever—and wherever—they find themselves in a bargaining situation.

But in order for any negotiation to take place, certain conditions must exist.

Is it Negotiable?

Contrary to the old saying, everything is *not* negotiable. You can negotiate the price of a used car, but no one is going to convince you to sell your children at any price (although there were times during the my kids' teenage years I wished childbirth was retroactive). Outside these two extremes, however, is a world rife with possibility.

When most people think about the kinds of things that are usually on the table in a negotiation, the first thing that comes to mind is *price*, which is almost always negotiable. But there are many other issues that are negotiable, depending on the circumstances. Delivery date, number of features, shipping, terms, services—all are negotiable at one time or another.

A Vested Interest

The party with whom you are negotiating must care about the outcome of the discussion. Without that, the best techniques in the world won't help you. If you try to negotiate with the tolltaker on the

Golden Gate bridge, for example, you're not going to get very far because he or she has no interest in the outcome. (I've tried this, by the way. The toll was a dollar at the time, but I pulled up and said, "Seventy-five cents, and that's my final offer." The tolltaker was not amused.)

Trust

Trust is critical in any negotiation. No matter how good your skills, you can't negotiate effectively if you and the person you're negotiating with don't trust each other. Both parties must command a level of credibility that often takes years to build. One way to gain trust is by making your words match your deeds. Get to meetings on time, hit your deadlines, and make accurate predictions. Over time, this will add strength to your position in most negotiations.

A further word on the subject of trust: I've worked with organizations in which management simply didn't trust the test group. They thought the test engineers were a bunch of Chicken Littles, overreacting to everything. I've seen organizations in which the developers looked at the test group as a police force, instead of a safety net. I've also heard it said that test engineers are second-class developers put in that group because they can't write good code. These are bad situations, and if they exist in your company, they are going to create problems.

I'm glad to report that this kind of prejudice is becoming rare. The test engineer plays a vital role in the software development process and is increasingly seen as essential to the success of the company. In an effective organization, the testing specialists are respected and the developers understand that the test engineers are going to save their butts from time to time by finding defects before the customers do.

Remember: There's no test tool, no process, and no technique that will help your company if you, its employees, don't trust each other.

Personality Styles

One of the first things I learned on my quest to acquire effective negotiating skills was that other people don't necessarily think the way I do. People, I learned, actually relate to work, data, and others in often completely different ways. What a revelation! Could it be that storming into the boss's office with a three-foot stack of software metrics wasn't always the best approach? Was it possible that direct confrontation actually turned some people off? Beginning with the human side of the negotiation equation, I came to realize that personality style was a critical factor.

Through a number of great teachers, writers, and the school of hard knocks I learned that there are many different ways to characterize personalities. But I'm sure you'll recognize your bosses, subordinates, and co-workers in the following four categories:

The Director: "Book 'em, Danno!"

Directors are usually found in management. The most important thing to people of this particular personality style is the bottom line. These folks are alert, very competitive, and they like to win. Being first is very important to them. They have the ability to juggle many tasks simultaneously, but they don't have the ability to focus for very long on any one thing. They tend to have superficial knowledge of many

things, but little deep knowledge of anything in particular. They like things to be measured, so the software metrics are very important, but they are action-oriented, so data are important only as they relate to outcomes. Jack Lord from the television show “Hawaii 5-0” is an example of a Director.

The Thinker: “That does not compute”

Thinkers are very different from Directors. These are process-oriented people. The most important thing to Thinkers is the analysis and logic used to come up with a particular conclusion. What counts is how you got your answer. They are great problem solvers, but they are very disinterested in the eventual result. They are detail oriented, methodical, and precise. They prefer working alone, and they love to gather data—and they’re very hard to convince unless they observe your point personally. You see a lot of Thinkers in the software industry. Data from *Star Trek* is an example of a Thinker.

The Socializer: “Hey everybody, let’s put on a show!”

Socializers are characterized by an emphasis on interactions between people. They love to be around people, they like teams, and they have a lot of energy. They are often very flexible, readily going with the flow. They tend to be highly visual, conceptual thinkers. They like to begin conversations by finding out what people do socially, and then move into business. They tend to feel uncomfortable with people who go right for the bottom line. Think of Mickey Rooney in the old Andy Hardy movies.

The Relator: “Can you say neighbor? I knew that you could.”

Relators are rarities in the software development world. This type is more concerned with the feelings of the people they work with than the job they’re doing. They are more interested in emotion than data. They tend to be empathetic, and they’re often very good listeners. They like personal contact, face-to-face contact, and physical contact. They tend to be reliable, patient, relaxed people. Mr. Rogers is a good example of a Relator.

Understanding these four personality types can help you to become a more effective communicator, which is a vital first step in becoming an effective negotiator. I also suggest taking the time to decide which category most closely fits your own personality style. (It might not hurt to get some outside feedback here.) You must know yourself to understand the chemistry you create when you interact with other types in a negotiation.

Let’s suppose you’re a Socializer interacting with a Thinker. You might tend to talk about the progress you’re making, the milestones you’ve passed, the excitement among the employees about your progress. You might even want to organize a celebration. The Thinker will ask what you mean by “milestone,” how you are measuring your progress, who reviewed your progress, and who signed off on it. Pretty soon the Socializer is discouraged and just wants to throw the whole project into the dumpster.

If you lean toward the style of the Relator, you might go to a Director and talk about a morale crisis. Everyone is depressed about the fact that they don’t have enough time to get things done. A couple of people are doing their résumés, and others are talking to headhunters. People are ready to leave the job. The Director won’t understand why any of this is important.

If you're a Thinker, you've probably collected tons of data— metrics, hours accumulated, projections—to support your contention that you need a longer testing schedule. Armed with this material, you march into the boss's office. The only problem is, he's a Director and all this detail is just going to drive him nuts. He's only concerned with one thing: the bottom line.

Based on my experience with these personality types, here is my model of modern corporate America:

* The people making the decisions tend to be Directors. They're focused on bottom-line results.
* The people responsible for the work tend to be Thinkers. They're so caught up in the details they can't see the bottom line.
* The people coming up with the product requirements, the marketing people, tend to be Socializers, who just don't care much about the details.

If this distribution of personality types is accurate—and I believe it is—then the most common interaction in the testing world is between Thinkers and Directors, which points up one of the biggest problems in our industry: Software developers and test engineers don't know how to present things in a bottom-line way. Yet I've found that when they begin talking about things like return on investment or the cost of inadequate testing—when they begin speaking the language of the decision-makers—people sit up and take notice. We cannot continue to bombard Corporate with data and expect anything to change.

Direct and Hidden Needs

Beyond personality styles, effective negotiators are cognizant of the needs, both direct and hidden, of the other party. Direct needs are obvious. Both the need to get the product out as soon as possible and to do it as cheaply as possible are examples of direct needs.

Hidden needs are trickier. I'm talking about the subtext of the human drama lying beneath the surface of any negotiation. The other person is probably unaware of these covert desires himself. The most effective negotiators develop a kind of radar that steers them though this shadowy space.

A list of some of the most common indirect needs could include the need to be right, the need for control, the need for prestige, and the need to win. These types are the easiest to deal with, all you have to do is tell them that they are right or that they win up front, or let them win or be right about some points, and then proceed to negotiate hard on the key issues. Remember, they actually need to hear the words "you win" or "you are right;" until they do, they can be intractable.

Sometimes the hidden need is about ego. The other person went out and made delivery-date promises without adequate information, and now his or her reputation is on the line. I've seen people take a position to avoid personal conflicts, to feel like they're doing a good job by making the deadline, to give the customers what they want. These types are passive/aggressives and "pleasers." They agree to anything face to face to avoid conflict, and then proceed with their own agendas behind your back. My best suggestion here is to call them on their game by pointing out diplomatically their incongruent words and actions. Attempt also to see things from their point of view; their motives are often noble. Create a non-

confronting comfort zone in which they can feel safe discussing their real concerns. I've even seen people act out their personal dysfunctional family dynamics in a negotiation (suddenly I become their overly critical father and receive 20 years of stored up rage). While it may be smart to retreat to safety in extreme cases, you might counter this behavior by telling them what you see. You might say, for instance, "I can see that your face is red, you are shouting, you are pounding your clinched fist on the table. This seems like a strong response to my asking you to discuss the testing schedule. What is really going on right now with you?"

This kind of beneath-the-surface stuff many not seem very professional, but it's very human. Effective negotiators are always on the lookout for hidden needs, and they adjust their strategies to accommodate them. Remember: The easiest thing in the world to negotiate for is your opponent's needs. If I'm negotiating for something I want that seems not to be in your best interest, you're going to resist me. If I'm negotiating to get you something that you want, something that's in your best interest, you're not going to resist so much.

What I'm talking about here is the strategy of going for a win-win outcome. It's always best if you can get what you want without doing so at the expense of your opponent. In many, if not most, bargaining situations, when one person loses, everybody loses in the long run.

Be Prepared

Despite the emphasis in this article on the techniques of negotiation, let's not forget the value of good data in any bargaining situation. Going into a negotiation without having your facts straight is a waste of everybody's time.

Start by collecting data from previous jobs. You will want to find out just how long it took to test the last product, how many lines of code were tested, and how many test cases you ran. How long did it take to run those test cases? How many people did you use to run them? How much equipment was needed? How many times did you have to rebuild the software after you found defects and retested?

You might also want to find out how many service calls were made on the last product. What kinds of problems were they fixing? What was the cost of maintaining the department that answers those calls? How many bug fixes did you have to send out?

Ask, how did we decide how much time to allocate for testing? Did we simply say that it's the time between when development stops and we deliver the product? If so, how do we know that's enough time? Do we just say, whatever is left over, we use for testing?

You'll also want to get a feel for industry-wide averages. How many bugs per thousand lines of code should you expect to find. In my experience, that number ranges from four to eight for an independent test group doing integration and system level testing. How long does it usually take to find each bug? My experience tells me that it takes somewhere between two and eight hours to find each bug including the time needed to design tests.

Return on Investment

When what you're after is financial support, say, a new test tool or some additional employees, one of your best negotiating tools is hard numbers. Nothing works better to get financial people on your side than a demonstration of a genuine return on investment.

Let's say you're asking for a $50,000 test tool. Of course, the tool will be costing your company more than its $50,000 price tag. There's the preliminary research to make certain you pick the right tool, and once you have it, you still have to spend some time learning how to use it. My rule of thumb is to double the cost of the item. In this case, you'll be asking management to spend around $100,000. That's a lot of money.

The first hard number you'll need is the cost of a bug that goes out undetected in a product release. The current consensus is that the cost of repairing a fielded bug is between $4,000 and $16,000. But sometimes, the costs are even greater. I worked with a client in the medical software industry who had to place 30 field technicians in 30 hospitals for a month to find, detect, and repair bugs. When I managed development and testing groups for military applications, we would sometimes have to send people out on an aircraft carrier for two weeks to deal with a bug, or send someone halfway around the world to be airlifted onto a submarine in the middle of the ocean. Remember the "bug that didn't matter" in the Intel Pentium processor? Intel CEO Andrew Grove reportedly said that it cost his company about $450 million. Intuit released a bug in its popular MacIntax program in 1996, and the company reportedly set aside $450 million to cover any claims that might arise from IRS fines. That's just two bugs and we're getting close to the billion-dollars mark.

Barry Boehm, in his 1981 then groundbreaking book "Software Engineering Economics", has suggested that the cost of a bug increases almost logarithmically as it slips through the different stages of software development.

There are a number of tools available for estimating the cost of bugs released into the field. They range from simple tools using Lines of Code, Function Points, or Feature Points and Risk Factors to much more comprehensive tools that contain large data bases of previous project costs and details like the office square footage per employee. Cost of these estimation tools range from free on the Internet to several thousand dollars. The costs involved in repairing a bug could include:

* Calls to your company's service desk.
* Collating each problem with similar problems and reporting them the quality assurance group.
* Setting up the data and preparation necessary to duplicate the problem to see whether it is, in fact, repeatable.
* Test reports and demonstrations to the engineering group.
* Looking at the code to find and isolate the problem.
* Designing a repair or fix.
* Testing that fix.
* Integrating that fix into the build.

* Regression testing the fixed build to make sure the fix didn't break anything that used to work.
* Coming out with a new version of the product and redistributing it.

When you begin putting together all those labor hours, the numbers add up quickly. But for our purposes here, let's go with the lower consensus number: $4,000.

There is even a gadget called a *bugometer*, which allows you to enter the real labor rates of your people, and the amount of time they spend repairing a defect. With it, you can estimate your own bug costs with a fair amount of accuracy..

Next, you'll need to calculate the number of bugs you'll have to fix to pay for your new testing tool. In this example, we divide the $100,000 cost of the new tool and associated training by the $4,000 cost of an undetected bug. To justify this expense, your tool will have to find 25 bugs you wouldn't have detected without it. If I know that we have 50,000 new lines of code, and our previous history or industry standards suggest that there are about five bugs per thousand lines, then I know that there are 250 bugs to find in this product.

If what you're buying is an automated capture playback compare tool, you may realistically claim that you will run it 24 hours a day, seven days a week, tripling your testing hours and finding even more defects.

So if you believe that the new $50,000 tool can find at least 25 bugs in the next 12 months that might otherwise have been missed, you have justified the tool with a return on investment of one year. If you really want to impress the finance folks, you could even include a cost of money calculation stating that dollars spent now for the tool are cheaper than dollars spent a year later fixing the defects. A dollar spent later will cost more because of inflation, which admittedly is only 3-4 percent in the U.S., but boy will you look smart in Israel or South America where inflation can run at 200 percent and higher.

Let's say you want to install an inspection system. Now you're not dealing with a physical tool, but the principle is the same. You'd like to bring five people together to form a peer group inspection team. Generally, in Silicon Valley, I've found that peer inspections take about 20 hours (five people times four hours). The team members have to read a syllabus or attend a briefing meeting to understand the code or document being reviewed. They have to spend one or two hours inspecting the document. Then they have to meet for an hour or two to discuss their findings. Assume an average labor rate of $100 per hour, which means the inspection will cost around $2,000. If your team finds even one bug during an inspection ($4,000 saving), you have doubled the return on your investment.

Or, you could look at it another way: Let's suppose that I know from our past history that it takes six hours on average to find each bug. Then I also know that I'm going to need about 1,500 hours of test time. If each of my test engineers can work roughly 160 hours per month, I know I'll need about four of them to complete the testing in three months. There are other factors to consider, but you get the picture; you're trying to develop a *rational* schedule.

This kind of bottom-line oriented hard data is invaluable in any negotiation, but don't forget to consult your gut, too. Your instincts are very important. What does your gut tell you is likely to happen on this project?

Frankly, I had little luck in my career soliciting any support that involved company expenditures until I learned to calculate a return on investment. But when I've got my numbers together, I've found, contrary to popular belief, that financial people are ready and willing to give me what I need—as long as they have the numbers they need. Once I learned how to justify a return on investment, it became easy to win the support of financial people and get the budget that I needed.

The Opposing Argument

Finally, as you're putting together your argument, take some time to prepare the *opposing* argument. There's no better way to prepare yourself for a negotiation than by anticipating clearly your opponent's point of view.

Tricks of the Trade

Finally, here are a few specific techniques I've picked up on my quest to squeeze more time out of the schedule for testing. The best techniques generally are awareness and sensitivity, but sometimes you need a trick or two. Here are some of my favorites:

The Cool Approach

"Okay, boss, if you want to ship a buggy product, I'm not going throw myself in front of the delivery truck. But what I'm going to do is report to you how buggy it is, and what the costs and consequences are of shipping that product without adequate testing. If it were me, I know what I'd do, but you make the call."

The idea here is to remain calm and collected, no matter what. Never get emotional and beat your breast; it just doesn't work. This technique works well with both Directors and Thinkers if you have your numbers together.

The Colombo

"Okay, okay, now let me see here... Normally, we're supposed to do this software in six months, but it took us 12 months this time, and we're really not done yet. And your strategy for getting this project to market on time is to cut the test schedule. We've doubled the amount of features, we've tripled the amount of lines of code, but you want to cut the time we allocate to test. I don't get it.

This is one of my favorite approaches, and I've found it to be very effective in numerous situations. It's obviously named after the character Peter Faulk played on the television series. If you'll recall, he always got his man.

The Reversal

"All right, you're not going to give me enough time to test the product well, so let's just not test it at all. I mean, why bother? The customers are going to hate it. It's going to be unacceptable anyway, so

why spend a couple months and all that money to come out with a product that's still buggy as a Roach Motel? We might as well just ship it when the developers think it's okay."

This kind of reverse psychology can be very effective. It gives the decision makers a chance to see what they're doing when they cut into the test schedule.

The Participation Strategy

Sometimes you have more negotiating leverage when the numbers are on your side. Try setting up a group that will have the power to decide when the product is ready. "We want to work with you Mr. Developer, Mr. Marketing, Mr. CEO. So we've established a team to decide when we really think this is ready to go. We're going to tell you what we think the risks are, and then together we're going to decide what to do and when to ship it out."

Using this strategy, you might want to establish some exit criteria, including things like, we've run all the test cases and the product passed them all; all the major bugs are out of the product; letting it run for a week without crashing; letting some of the customers try it out and getting some positive feedback; asking the members of the test group how they feel about the product and getting positive feedback.

Limited Authority

One thing you never want to do is blurt out the answer (like a schedule date) to a question from management without thinking it over carefully. There have been many times in my career when I've been pushed against the wall by a senior project manager asking me to tell him when we would be finished testing. Every time I opened my mouth with even the most reasonable answer, I regretted it.

One way to get out of a spot like this is to say simply that you have to check with someone else in the group. You can even get away with this if you're the responsible test manager. They do this very well in car lots. You're ready to buy a car, you're all set to make the deal, but the salesman has check with the sales manager.

Never give a snap answer to this kind of question. Always say there's someone you have to check with, and then get back to your interrogator when you've thought your answer through and discussed it with all who may be affected. This gives you a little time to think before you speak.

Bracketing

Everyone has heard of the bracketing technique (Donald Trump illustrated it in his book, *The Art of the Deal*). If you need six months, you ask for eight because you know they're going to knock a few months off whatever you ask for. Another version involves including every possible test in the schedule, knowing that management will knock out at least a few of them. It's not so much a matter of padding as it is being particularly vigorous.

I'm a little uncomfortable recommending this negotiating technique, because deception is not something I want to encourage. When you're part of a healthy organization, when you have a solid relationship established, when you trust the people you work with, it's always better to be straightforward

and deal honestly. However, others are going to bracket *you*. When you know it's coming, remember that you have the option of bracketing back.

Horsetrading

The boss says flat out, I'm not going to give you any more time to test. You say, Okay, so how about buying me a test tool? Can I bring in two consultants? How about letting me use some contract workers? Can I take part of this job and give it to a third-party test group? Can we leave off some features in this release?

If you can't get the time you need, bargain for something else to help you get the job done in the time allowed.

Switch Hitters

You might not always be the best one to step up to the plate for a particular negotiation, either because of your personality style, or your history with the other negotiator. Why not send in someone else who has a better rapport with the other party. Maybe it's someone from your group who plays on the softball team, or drinks beer with him or her on Friday nights. If it seems ethical, take advantage of those relationships to strengthen your negotiating position.

The Salami

Many times you'll find yourself in a company or a situation where you just can't win it all the first time around. But there's almost always *something* you can take away from the negotiating table. Whatever that is, go for it. Take a slice today, and go for a bigger slice next time. Over time and several different projects, you'll gain trust and credibility, you'll develop your negotiating skills, and the data will pile up to support you. Eventually, you just might get the whole salami.

Whether we realize it or not, we all go through life negotiating. One of the things that determines our overall success is the degree to which we succeed at this fundamental task. For most of us, preparing for this isn't second nature, but it is a skill anyone can learn.

Giving a "Hoot" about the importance of personality styles and effective negotiating techniques has helped me be a more effective manager, software developer, and tester. Remember, unrealistic schedules are the number one obstacle to software product quality in most organizations. Your own innate, Owl-like wisdom can help illuminate the darkness of the software scheduling process.

###

THE AUTHOR:

Gregory M. Pope is the president and founder of Azor, Inc., a Silicon Valley-based company founded in 1993 specializing in e-commerce outsource testing and project management services, test automation and tools, development methodology consulting and training. Over the past quarter century, Mr. Pope has worked in a variety of capacities at virtually all levels of software development and testing.

He began his career developing software used to test jet engines and helicopters. He later worked in the defense industry, testing mission-critical software for military and space applications. Working in the private sector, he has developed and patented techniques for computer-aided testing. Among his inventions is the Ferret, a highly regarded software-testing tool manufactured and marketed by his company.

As a consultant and teacher, Mr. Pope has conducted hundreds of seminars for software development professionals throughout the United States, Asia, Canada, Mexico, South America, and Europe. In his thriving consulting, training, and testing practice, he has worked with many Fortune 500 companies, including Microsoft, IBM, Apple Computer, Sun Microsystems, AT&T, Eastman Kodak, DHL Airways, and Knight-Ridder, as well as NASA, the Pentagon, U.S. defense contractors, the Internal Revenue Service, and numerous foreign companies.

Mr. Pope is sought out to write articles on the subject of software testing for a number of industry publications, including CIO Magazine, *Computer Design*, *Industry Week*, *Computer World*, *Signal*, *Electronic Defense News*, *San Jose Mercury*, and *Software Maintenance News.* He holds a BS degree from Connecticut State University, an MBA from University of Phoenix, and is a member of IEEE.

GREGORY M. POPE

Gregory M. Pope is the president and founder of Azor, Inc., a Silicon Valley-based company founded in 1993, specializing in e-commerce outsource testing and project management services, test automation and tools, development methodology consulting and training. Over the past quarter century, Mr. Pope has worked in a variety of capacities at virtually all levels of software development and testing. He began his career developing software used to test jet engines and helicopters. He later worked in the defense industry, testing mission-critical software for military and space applications. Working in the private sector, he has developed and patented techniques for computer-aided testing. Among his inventions is the Ferret, a highly regarded software-testing tool manufactured and marketed by his company.

As a consultant and teacher, Mr. Pope has conducted hundreds of seminars for software development professionals throughout the United States, Asia, Canada, Mexico, South America, and Europe. In his thriving consulting, training, and testing practice, he has worked with many Fortune 500 companies, including Microsoft, IBM, Apple Computer, Sun Microsystems, AT&T, Eastman Kodak, DHL Airways, and Knight-Ridder, as well as NASA, the Pentagon, U.S. defense contractors, the Internal Revenue Service, and numerous foreign companies.

Mr. Pope is sought out to write articles on the subject of software testing for a number of industry publications, including CIO Magazine, *Computer Design, Industry Week, Computer World, Signal, Electronic Defense News, San Jose Mercury,* and *Software Maintenance News.* He holds a BS degree from Connecticut State University, an MBA from University of Phoenix, and is a member of IEEE.

SAM GUCKENHEIMER

Sam Guckenheimer is the senior director of marketing Rational's Automated Testing products. In this role, he is responsible for the product direction and implementation of Rational's software testing products. He joined the engineering team at SQA Inc. in 1995 as director of technology integration and moved into his current position when SQA merged with Rational in early 1997.

Sam has held several marketing, engineering, and general management positions in US and European software companies over the last fifteen years. At Rational, Guckenheimer has spearheaded the integration of the load testing products, the development of Web server testing technologies, the internationalization of SQA Suite, and the introduction of OEM versions of these products. Prior to joining SQA, he spent six years with Softbridge, Inc., ending his time there as managing director of Softbridge Capital Markets, a subsidiary based in London, England.

A Phi Beta Kappa graduate of Harvard University, he is a frequent speaker on software development and application topics at industry conferences and seminars, and has spoken as a guest lecturer at the management schools of MIT, Harvard, and Yale.

ANDREW L. POLLNER

Andrew L. Pollner is President and founder of ALP International Corporation, a leading organization in the area of test process improvement and in the use of test automation tools since 1993. As an early adopter of test automation and quality assurance tools, Mr. Pollner has built a consulting practice serving major financial, insurance, healthcare, and telecommunications corporations ensuring their successful implementation of test automation tools and surrounding processes.

Mr. Pollner regularly speaks at major national and international testing conferences, including: STAR (Software Testing Analysis & Review), EuroSTAR, USPDI, Mercury Interactive Worldwide, and has published articles on software testing, and automation.

HARRY ROBINSON

Harry Robinson is a software test engineer with the Intelligent Search Test Group at Microsoft. He has a B.A. degree in Religion from Dartmouth College and a B.S. and M.S. degree in electrical engineering from the Cooper Union for the Advancement of Science and Art.

He was a software developer for six years before coming to his senses and switching to testing. Prior to joining Microsoft in 1998, he spent 10 years with AT&T Bell Laboratories and then three years with Hewlett-Packard. He is a long-time advocate and practitioner of model-based testing.