How to Deliver Quality Applications
On Time and Within Budget

By Vince Kellen

Presented to the Inprise Conference - Europe, London, UK, 10/27/98.

Introduction

Since the beginning, the computer world looked at software development and general IT deployment as an engineering problem. If you can blueprint it, it can be done. The problem with this approach is that even with all the methodologies and blueprints that have proliferated over the years, software projects, especially complex ones, are very hard to get done on time and within budget. Why is this? Are the methodologies inadequate? Are companies not following the methodology? Is something else afoot?

I am inclined to think that it is all of the above. First let's look at methodologies. Obviously their inadequacy does not have anything to do with their depth and breadth of details. Many software development methodologies have tremendous wealth of details both broad and deep. Are organizations not following the methodologies? Probably so. But there are many organizations that are very serious about their methodologies, follow them well but still have some problems delivering projects on time and within budget.

To see why you must remember that methodologies try to protect organizations from breakdowns in work activities. The methodologies attempt to catch these breakdowns at some point in the process so it can be corrected. Few methodologies attempt to prevent the breakdown in the process from *being introduced in the first place*. Those that do usually take a systemic approach to the solution rather than a personal one. The methodology seeks to put in place a procedure or policy or workflow to address to problem. It doesn't seek to address the various personal and intangible factors that precipitated the breakdown. Methodologies don't delve into those personal and intangible factors. That is in the realm of human behavior that only the individual, not the organization, should change.

The conundrum before us is that ultimately it is only other human beings and human culture that can change human behavior, not written policies, procedures and workflows. It is through human beings and organizational cultures that written policies, procedures and workflows are given life, not the other way around.

Is something else afoot? I think so. What is that something else?

The problem is that most organizations don't really know how to improve people's skills. Oh any organization can certainly improve its methodologies, especially the documentation of those methodologies. And certainly it can send people to all sorts of classes. But do these organizations really improve their peoples' ability to deliver? How do you improve your ability to deliver?

Discipline: Are You Battle-Ready?

It comes down to one word: discipline. First let's define it. Discipline is not buckling down, sucking it in, enduring pain, forcing yourself to do what you would rather not do. Discipline is not a negative thing. While discipline does involve a little bit of this, mostly it involves constructing an environment where people's natural instincts and skills for doing great things are encouraged. Think of discipline in two ways at the same time: having the ability to motivate yourself to accomplish great things on a regular basis and having the intelligence to alter your environment so that you are more likely to accomplish these things on a regular basis. When we think of great musicians or great athletes, we often think of discipline. Clearly they did not get where they are because they forced themselves to do something they didn't want to. They obviously enjoy it. But what do great athletes and great musicians do more than most of their peers that enable them to get to that next level? Practice.

Practice, practice, practice

In his treatise on the Roman army, the Jewish aristocrat, Josephus noted that the Roman army practice sessions were bloodless wars and their actual battles were bloody practices. In other words, he saw little difference between the two. One of the keys to teams that perform well is that practice sessions are treated with the same intensity as real battles. In software development, how many practice sessions do developers and team leaders get? Usually none. Instead they get thrown in a project right away after having a few days or weeks of training sessions. Their first battle becomes their first real practice session. I have never attended nor observed a training session that was even close to a real-life project. I rarely, if ever, get to see a real life project that can be treated as a training session. Usually there is little or no time or money for just a practice project.

How should we get around this? Simple. Everyone should practice, practice, practice - anyway they can. Find anything to practice with, even if it doesn't apply to a specific project. Got a group meeting coming up? Practice meeting preparation and management skills. Got an interview with a group of users? Practice small group communications skills. Want to learn a new tool? Develop a checkbook balancer program in it. Great teams use *absolutely everything* as a learning opportunity. What should you practice? All things technical, administrative and team oriented, which I will discuss later.

Repeat intensely

When practicing, it is not enough to simply do it once. Rather, the skill being learned must be repeated *with as much intensity as possible* at least a few times. Repetition without intensity does not promote learning. In order to improve the skills of your team or yourself, you must improve the rate at which the team or you learn. Studies have shown that when people get emotionally charged about what they are learning, they retain the information better. In fact, I contend that what separates masters in a discipline from journeyman is *simply their ability to get emotionally charged about what they are doing*, not their innate intelligence or skill. If you or your team is not absolutely emotionally riveted about the project or the job, improvement is highly unlikely.

Attend to All Small Details

Masterful teams and individuals have fanatical passion for all sorts of details, especially the ones that are core to their role. If the area in question is a bit of code, or the design of a user interface, or a data model, or a network architecture, teams that improve greatly eagerly delve into all sorts of little details. They do not consider any detail too small.

On the other hand, these teams and individuals do not get bogged down into too many details, especially inessential ones. Improving demands attending to details *one at a time* so that one detail is mastered before moving on to the next.

If you see your team passionlessly glossing over details, be concerned.

Re-Invent Administrative Rules

In order to get projects done on time, within budget and at a certain quality level, someone on the team must be passionate about administrative details as well. The problem here is that many teams are saddled with bags of administrative gravel that adds dubious value to the project at hand. But every project does have administrative details that are essential for successful completion, like project plans, time tracking, defect tracking and so on. For example, in American baseball, when a player at bat hits a home run, the player must touch all bases in order for the home run to count. The player simply cannot miss any base. While this may seem a trivial detail, *it is a rule of the game* that every player takes seriously. In IT, clever teams find ways of defining a methodology with rules that they can get passionate about, thereby ensuring that they master administrative details.

If you or your team is not passionate about those essential administrative details, invent new exciting ones to replace the old boring ones.

Make Good Use of Role Players

Many players on a team are often dissatisfied with their role and hence their performance diminishes. For successful deployment of complex applications, it is critical that this problem gets examined and resolved. Too often managers look the other way. In order to solve this problem, two things must happen:

1. Managers can recruit and hire team players who take pride in role playing.

2. Managers must learn to recognize team players' *true* skills and take advantage of roles that people, by nature, want to play, rather than force people into roles that by nature, they cannot play.

The best teams have players that love their roles, are good at it and get recognized for performing their role. Very rarely are large teams that create large applications managed with this sort of concern first and foremost on the manager’s mind. Too often the manager is concerned about budgets, schedule and other project problems that at the time, seem far more important than making everyone on the team happy. In addition, rarely is a team constituted by carefully considering the mix of role players that make for a good team. More often than not, good teams are brought together by accident. On the other hand, managers can’t run around forever pleasing malcontents. Each player must exhibit professionalism and perform their role as expertly as they can, regardless of the current state of affairs.

Pain and Avoidance

Projects that have problems often have a lot of pain associated with them. When faced with this situation, many managers respond by creating more pain and suffering, assuming that by getting mean and tough, discipline will be enforced. The following old saw explains it perfectly:

“The floggings will stop when morale improves around here!”

High performing teams realize that when things are tough, everyone must cultivate a positive, patient and optimistic outlook. By nature, people will avoid engaging in situations that are painful. When projects go awry and pain is increased, managers who increase pain further are unwittingly encouraging their team to disengage from the battle.

Great sports players love the tense situation. They are willing and eager to be at the center of things when everything is on the line. And when they are ready to perform, they are not thinking "Oh my gosh, everyone is watching me. What happens if I miss?" Instead they are thinking "This shot is going to go exactly where I want it."

On just about every project I have observed that is in a tense spot, most of the managers and team players *increase* their skepticism and *increase* their nervousness, precisely the opposite of what is required. When confronted with projects or games about to veer off course, great teams and great players see nothing but a great opportunity. While the great team and great player may not always succeed, *they are always confident that they can succeed.*

Optimism is required not when a project is running well, but when it is running amok.

Small Steps

Getting your teams to improve requires taking small steps. Masters in any discipline did not reach the level of mastery overnight. They did not experience one sudden rush of instant improvement in a single day. Most of them worked hard and slow, bit by bit. They achieved a level of mastery that to the beginner looks like an enormous gulf but to the master is just a lot of little tiny steps over time. Delivering quality projects on time and within budget requires a long-term vision that accepts small steps.

Know When to Reinvent the Team

From time to time all teams reinvent themselves, but usually only when they have reached rock bottom or will soon be falling to rock bottom very quickly. For example, if you know that all four key members of your team are about to leave, then reinventing the team might be in order. But for those teams that are reasonably well constituted and performing, incremental improvements are much better. When top golf players are having problems with a portion of their game, they usually do not decide to reinvent their swing. Instead they make small adjustments. Great golf players are especially adept at always making small adjustments. So it is with great teams.

Persistent, Consistent Effort

There is one idea I consider the most important for teams that wish to produce great things regularly. This is the idea of consistency and persistence. Great teams are consistently persistent, not just for a day or a week, but week in and week out. They always dig at problems and look for solutions. Moreover, when not confronted with a problem that demands persistency, they perform their job consistently. They always keep on going. They are persistently consistent.

With consistent, persistent effort, small changes can be accomplished one right after another, day in day out without ever missing a day. After a year, teams and individuals that consistently persist amass an amazing amount of skill and knowledge. Just as it is important to be emotionally charged about what you are doing, it is important to be consistently charged day in and day out. The great ones do this. If you are committed to taking small steps, you must consistently make these steps every day.

Kick it Up a Notch

The notion of persistence and consistency must be taken a step further. Great teams and individuals want to accomplish more. After mastering one skill, they want a bigger one. They are always seeking to "kick it up a notch." Those individuals and teams that are bold enough to do this learn the necessary skills that enable them to handle the mega-project. They are not content with small potatoes. They want to change the world.

Taking it up a notch not only applies to technical challenges, but also other things that important projects require: public speaking, running meetings, training, mentoring. All the various "teamwork" skills that are often lumped on a manager's shoulders are tasks that every team player can perform as well. Great teams don't just have one manager. Often they have team players that possess key parts of great management skills. This means that the team leader can rely on others to communicate to and mentor others on the team. When the team is constituted with leaders who have a positive attitude that can "kick it up a notch," other team members pick up on this and begin to "turn up" their game. I don't want to call this internal competition, because it isn't. Instead, I like to call it a contagious attitude. Since human beings are social animals, we tend to do things that others are doing too. Team members usually raise their skill level to meet team leaders' expectations.

Take Advantage of Small Skirmishes

Great generals throughout history rarely risked the lives of their troops on great battles until they had tested them and trained them thoroughly on small skirmishes. In fact, many a soldier has been confused as to why the general was ordering these seemingly pointless and small skirmishes. When your troops are green and fresh, as it is most of the time in the world if IT, it is vital that team players get to sharpen their skills on projects where failure has little or no lasting negative impact. But in order for the practice to be real, team players do not necessarily need to know this.

Over the years, we have always had smaller internal projects in which the rules of project management are vigorously applied. We use these projects to train and test new people rather than doing it on the big project where failure can be demoralizing. Even when you detect great talent in team players, you must always use small skirmishes to test and train them.

Avoid the Revolution (Unless You Are Fully Prepared)

Great teams only embark upon really difficult projects unless they are fully committed and prepared. In the words of Tom DeMarco, these are the so-called death marches; projects that will have casualties of some kind along the way. Death marches tend to have an air of futility to them, they are often not very exciting or exhilarating projects. Usually only organizations that are ignorant of how information technology projects should be done attempt death marches. And usually you don't find great teams in ignorant organizations. If you want the project done well on time and within budget, don't take on the death march.

One type of death march project, which I will call the revolution, doesn't look like a death march at first because it is cloaked in a veil of glory and significance. The organization's future success is often tied to these projects. These projects have an "all-or-nothing" feel to them. Either the project will succeed, or all will come crashing in around us. If you are taking on a revolution, make sure your team is well trained, well motivated, has had plenty of small skirmishes behind them and is extremely loyal to the cause. Otherwise, the counter-revolution will crush you.

Everyone Should Look For Replacements

While each team member must be willing role-players, each team member who wishes to progress in their skills should have one thing on their minds: "Who will replace me?" Team managers should look for replacements for everyone so that new teams can be created, so that role-players can expand their skills be moving into new roles and so that new people can be quickly integrated into the team.

In order to meet the demands from the field, the IT group must be prepared to expand its numbers of teams (or work with 3rd parties here), and deal with the need to challenge and reward top performers who will very likely leave as soon as things get boring. When everyone looks for replacements, the team can survive into the future and continue to deploy great applications.

Customer is King

While certain fundamentals regarding team building are important, how the customer perceives the team is essential. Very often success is not measured in purely numerical ways, as in a project being done by a certain date within a certain budget. The communication between the IT group and the rest of the world is crucial. It is in this dialog between customer and provider, that many intangible elements enhance the relationship. The problem here is that many IT professionals are in their positions because they are excellent in dealing with machines, not people.

These team members need to be educated in how to interact with customers in a positive way. Or they must be shielded from the customer and a customer liaison should interact with the customer. Perhaps this is why the car salesperson sells us a car rather than the factory team that built the car. *(Although some would argue that perhaps the factory team should be doing the selling!)*

Never Criticize the Customer Behind Their Back

IT professionals are often notorious for calling users idiots. The following statement takes this name-calling to its logical end:

"If it weren't for the users, the application would run just fine!"

Or the even more extreme, software chauvinist view:

"If it weren't for this darned computer, the application would run fine!"

Obviously the reason we are in this profession is that users pay us to do our jobs. We can't do our jobs without users and computers. But there are other more subtle reasons for never criticizing the customer. We often overly criticize what we are ignorant of and what we are ignorant of we are hopeless to improve. The path to improving an application for a customer is paved in *understanding.* When the IT group fully understands the customer, there is often no distinguishing between the two. When this occurs, the IT group is an ally of the customer, not their adversary. Great applications become possible, because another foe (the customer) has been removed from the field of battle and made an ally.

One of the oldest rules of warfare is to use alliances wisely. By doing so, nations avoid accumulating too many enemies. IT professionals often proceed absolutely unaware that criticizing the client can make the client the enemy. And when it comes to building applications, who needs more troubles?

Remember, the enemy is not the customer, but the complexity of the project. Make an ally of the customer and together attack the real enemy.

Fight for the Customer's Best Interests

If we take the relationship between the IT group and the user to the next level, not only should we ally with the users, we should fight to protect their interests. When engaging in application development, we should constantly ask ourselves: "What is in their best interests?" Sometimes our understanding of their best interests conflicts with theirs. Often the IT group doesn't do a good enough job articulating their position.

Consider the case of a sick man, who, upon seeing a doctor, asks to have his appendix removed. The doctor kindly tells him that the problem is not with the appendix, but with the spleen and will require surgery to correct. The man insists that the doctor remove his appendix. What should the doctor do? Perform surgery on the spleen of course!

In the world of IT, we are often confronted with users wanting the wrong things fixed. When this occurs, we must do as the doctor does and look after the long-term and short-term health of our client.

As an aside, this also means that we should know when to say no, especially when confronted with a death march.

Tell the Bad News First

Don't wait until the end of the project to tell the customer the project won't be done on time. The best place to tell them the bad news is *as soon as the project starts!* But how do we know that early that the project will be doomed? This is where having an excellent project estimation method and project-tracking system will help. The systems should provide two basic metrics: How much gas is in the tank and how fast is the team going. The first measure, how much gas is in the tank, tells you *how big* an application is. The second measure, *how fast is the team going*, tells you when you will reach your destination. If you can get a reasonable fix on size of the application and the rate of completion, you should be able to tell your customer the bad news first.

I am constantly amazed by the sheer number of IT professionals who begin projects not knowing either number. Imagine the following conversation between a car purchaser and a car salesman:

A couple walks in to an auto dealer and says they want to buy a car. The sales person asks "What kind of car? How many doors?" The couple reply, "Oh we're not sure, maybe two, maybe six, somewhere around there." The sales person looks puzzled. He then asks, "What kind of wheel options do you want?" The couple reply, "We think we need seven wheels, but it might be ten, unless our children want to help us in the decision, in which case it might be three." Even more puzzled, the dealer asks what sort of engine options do they want. "We're not sure if it is a 6, 8 or 12 cylinder engine and whether it will be diesel, gas or electric," the couple respond.

"Oh and by the way," the couple finish, "Can you give me a price quote of plus or minus 5% on all that with guaranteed delivery in three weeks or we get a 50% reduction in price?"

Never in a million years (well maybe in a million years!) would we expect a car built within any reasonable time frame to accommodate these options. Yet in the world of IT, this sort of budget planning occurs *all the time in all sorts of organizations across the globe.* It really boggles the mind. What we try to do in cases like this is to construct a straw man which will be a basis for documented deviation. Here's how our sales person would respond to these questions:

“Since you are uncertain of many key items, I will give you a cost and delivery schedule based on the following options: 4-door car, 4 wheels, 8 cylinder gas engine. Should you decide to change your mind at any point in the construction process, we will document your change and let you know of the cost and schedule impact, before we make any changes. Depending on your change, the impact on cost and schedule may be significant. For example, if you decide, after we have built the engine, that you would like another engine, that will be a significant impact. If you decide that you simply want the car painted another color, that would not have a significant impact. This way you can completely aware of total costs and final delivery schedules.”

With a statement like this, we have defined the solution and started giving the bad news *right away*: If the system changes materially from the description, total system cost will rise. More often, however, customers come up with high-level business requirements and want the IT group to tell them how much it will cost. The conversation goes like this:

“The car must allow people to enter and exit, accelerate and decelerate, park, brake and turn. People must be allowed to enter quickly and exit quickly and windows must be transparent.”

Based on this specification, it is uncertain exactly what should be built. If we can’t come up with a specific list of features (in IT terms, lists of forms, reports, tables, web pages, etc.), we can’t really give a good quote. However, in these cases I do two things: give a powers of 10 quote and give a price on performing a more detailed requirements analysis and high-level application design.

A powers-of-ten quote goes like this:

“Based on your system description, I can’t give you a good quote. But I can tell you that the system is not less than $1,000, not less than $10,000 and not less than $100,000, but probably between $100,000 and $1,000,000.”

While this might seem a ridiculous way to quote systems, this technique does have advantages:

- It let’s you do a “pulse-check” where you can watch your customer’s reaction as you slide up the scale. This is useful if your users aren’t forthcoming with the budget they’ve got and want you to prematurely commit to too low of a number. When you see their veins popping out of their necks and forehead you can back down to the lower number and start to discuss what they can afford.

- For some customers, knowing that a problem falls within a power-of-ten range can be very comforting, especially if their internal estimates are much wider or completely unknown. Many non-technical people have no concept of how much custom software costs or should cost.

In order to tell customers the bad news first, you must have a good software estimation and project tracking methodology. Notice that so far, the word methodology has had a limited, but vitally important place in this discussion.

Customer Satisfaction Formula

In order for a project to be a success, it is not enough for the team or the IT group to call it a success. It is too easy for that to happen and for the customer to assume you are shouting victory in the middle of a retreat. The customer must call the project a success. In order to get the customer to call the project a success, get them to define what makes for a successful project and get those success factors documented

first thing. Success factors might be purely numerical, as in ROI, payback, cost reductions, etc. Or they may be entirely intangible, as in "We define success as the CEO smiling." Whatever it is, document and work to meet it.

In addition, pay close attention to the customer satisfaction formula:

$$\frac{\sum_{x=1}^{z} \frac{D_x}{E_x}}{z} = 1$$

In this formula, D is Delivery in terms of the application, its features, size and appropriateness. E is the users' expectation with regards to D. Clearly if D exceeds E, things are pretty good because we have exceeded our customer's expectation. Most of us are taught to exceed our customer's expectations. I tend to have a more refined approach to the problem. This is apparent in the rest of the formula. Here, the ratio of D/E is summed over a series of engagements (x refers to a given engagement and z is the total number of engagements) and divided by the total number of engagements. This is simply an average of the sums of D/E. Consider how this plays out in a series of engagements where each engagement is represented by a ratio of D/E:

1/1 + 2/4 + 4/4 + 3/3 = 3.5

3.5 / 4 = 0.875

The series of engagements averages to 0.875. In this formula, we must strive to deliver a D/E ratio of 1. If we don't and we always over-deliver, users' expectations (E) will naturally rise for the next engagement and this may catch us unaware. Expectations can rise farther and faster than they can fall. In fact, expectations rarely fall for too long. If we continually fail to meet our users' expectations, we will be replaced by someone who can. The moral here is that we must become expert at controlling E as well as D. At first this might seem startling because all of our IT training attempts to make us expert in controlling D, and rightfully so, and never really talks about E.

My advice is simple: worry far more about controlling E because E usually requires *no money or time to adjust!* If you need to increase D, you will need more money or time. All E requires is someone who is on the same wavelength as the user and who can communicate effectively about what is going to be delivered and when.

Also keep in mind that E might not have anything to do with that very thick, very detailed, analysis and design document you just completed and the user signed off on. It has more to do with understanding the users' definition of success. If the user can't define success, you can't get a handle on E and you can't really control customer satisfaction.

If you can't get a handle on expectations (real, not just documented ones) your chances for a successful project are diminished.

Over Time, Deliver More For Less

The world of information technology changes quickly and rapidly, perhaps more quickly than clothes fashions. Despite this maelstrom of changes, try to become more efficient and productive in producing applications. This is where having an object reuse methodology and a software construction process methodology can help.

I would hate to think that 30 years from now the same applications take the same amount of time to construct as they do now. If we don't improve our productivity over time, that would be a disservice to the business world and will guarantee our obsolescence as professionals.

Live, Breathe, Drink and Eat Quality

Great software is produced by people who have a passion for detail. This passion for detail is the basis for improving quality. Many of us have been in organizations that have defined quality improvement processes or methodologies. These programs do nothing more than provide a channel for this passion to be enhanced and directed in a coordinated way. Without the passion and the cultivation of the passion, a quality methodology is an empty shell. Team players, if they are passionate about quality, *must be passionate about any quality program in place*. If they are not, then it is not a quality quality program (pardon the redundancy). Make sure to have a quality quality program.

Your Back End is Your Front End

When proceeding down the quality path, you can get so insanely focused on the process of building software and designing ways to avoid the introduction of breakdowns and defects that you lose sight of where quality counts. Quality matters most when the customer experiences the software in production for the very first time. I've seen organizations that have had good quality programs but neglected the customer experience during deployment. In these cases, a great piece of software was ruined by a bad deployment experience.

Software deployment is like giving birth: the longer the process, the greater chance of something going wrong and the more painful the experience. Or, to use another metaphor, deploying software is like sending your child to kindergarten for the first time. You love your child dearly and you hope the child will make a good impression but very often, the first day is a very rough one. *And just because the first day was rough does not mean the entire year will be rough as well or that the child will be a nasty adult.* Nonetheless, you must do everything you can to make sure the first day goes well. In addition, you must have a process that can quickly correct defects.

To the development team, the process of deploying software may seem like a boring denouement, especially after a long and grueling development period. But it is not the time for attention to slacken or vitality to wane. Just the opposite is required.

Rationalize and Prioritize the Irrational

It is during the beta cycle of software deployment (you do have a beta cycle, don't you?) that users and clients begin to get rattled. Beta cycles are designed to get users to cause the system to crash so that *as many defects as possible can be found.* The following statement always constantly amuses me:

> "I don't know about this software. It has so many bugs in the beta cycle."

The problem with this statement is that it critiques the beta cycle for precisely what the beta cycle is designed to do: find as many defects as possible. The real issue with testing and beta cycles is not how many defects are found, but how quickly they are removed and the rate at which new defects are no longer found. And besides, if you associate finding defects in the beta cycle with pain and punishment, your team will find a way to frustrate successful project completion. They may overstate delivery schedules, under-engineer functionality, refuse to take risks or quit their job. And of course, there are plenty of good technical people waiting to fill their shoes, right?

The better thing to do is to rationalize and prioritize this process (and any process that causes irrationality). The first thing to do is measure what is causing people grief. If it is defects, measure how many defects are found, removed and the rate of removal. If it is user-friendliness, measure mouse clicks, keystrokes, mouse travel distance, conduct user surveys and so on. If it is performance, break it down to a series of timings for critical pieces of the application. *Don't let negative adjectives and adverbs get by without measurement!* This forces the discussion to be more logical, empirical and scientific. And that can be managed more easily than 50 end users and 10 developers running around shrieking. It forces all parties to speak a common language, (numbers) when discussing problems.

Great teams, players and managers never engage in finger pointing and placing blame. Instead they quantify, diagnose and develop new approaches. They don't let any negativity lead them away from finding solutions.

Over-repair the Damage

People are human and despite all the quality programs in the world, people will still make mistakes. However, because of the pain and suffering and negativity that mistakes often bring, team players try to *sweep the problem under the rug*. Most often, team players do not aggressively and *cheerfully* deal with the problem. Who likes to do anything painful? What often happens is that the processing of correcting the mistake is also botched and the customer now sees two mistakes, not just one. In many cases, I have seen errors trebled and quadrupled as team players botch first, second and third attempts at repair.

When mistakes occur, they should be looked at as golden opportunities and *over-repaired.* By over-repaired I mean that the team players should handle the error differently than they way they handle day-to-day business. After all, when a celebrated guest comes to visit your house, like most hosts, you clean the house, bring out their finer china, dress sharply and so on. I try to treat mistakes as honored guests that are given the utmost in attention. By doing so, you can ensure that the damage is resolutely handled and completely repaired. The customer gets to see you demonstrate your skill and can more easily forgive the mistake.

Remember, defects beget defects.

Alignment of Interests

Most organizations don't align interests well. Perhaps this is just not in the business world's mindset. For developing software on time and within budget, especially in today's world of a limited number of talented people, alignment of interests is crucial. Great team managers are great at aligning interest. When Julius Caesar crossed the Rubicon River and declared war on his fellow Romans, of which the outcome was very much in doubt, he did so with a willing army. His interest and their interests were aligned. In order to successfully tackle important projects, your team's interests must be exactly aligned with your interests and the organization's interests.

Reward Employees for What You Want

Fixing this problem is remarkably simple. Reward employees for what you want to occur. Most systems of reward are often too abstract, not compelling or just in plain contradiction with the goals of the organization. If you want fewer defects in software, reward teams for delivering good software. This is not the same as punishing those who do not. If you want more productivity, reward teams for being productive, don't just march up and down the galley yelling at everyone to row faster. If you want teams to pay attention to details, build it into the compensation plan.

The problem with this is that many organizations have compensation plans set by people who know nothing about delivering software on time and within budget. But consider the following example. The Chicago Bulls, a professional basketball team, had a player, Dennis Rodman (often called a malcontent and "head case" by other teams), who is notorious for getting thrown out of games for bad behavior, for missing practices and for many other disruptive antics. For the 1997-1998 season, Bulls management wrote a contract that specifically measured and classified these "disruptions" and rewarded Dennis nicely for meeting goals that reduced the numbers of disruptions. Guess what happened? In 1997-1998 Dennis Rodman had far less disruptions, a much bigger paycheck and the Bulls won the basketball championship. Funny how that works?

Organization Goals = Employee Goals

While short-term alignment of interests is crucial, bigger and more long-term interests must be aligned. For example, many organizations have a goal of rising sales, increased profits and increased productivity. For IT workers, this usually translates to more work in less time, which is usually not an employee goal. Organizations must find ways to make sure that their goals can be translated into corresponding goals for each employee.

For example, how should the IT manager accomplish the following goals?

1. Develop Java programming skills
2. Reduce defect counts
3. Improve turnaround time on help-line calls
4. Find 3 more talented programmers

Simple. The manager should get each member of the team to tackle one goal each. Delegating organization goals downward does two things: it aligns interests and makes people lower in the organization responsible for making the organization perform.

In the right climate, delegation means empowerment.

Reward Customers for What You Want

Too often, in IT, the customer is viewed as the enemy. All IT shops should have the exact same thinking about their customers as they do their employees. How can we reward customers for helping us meet our goals, which ironically, means rewarding customers for meeting their goals? Since IT groups must not only serve their customers but must also bravely lead them through a thicket of tangled technologies, it is important to think about customers in the same way. How can they be cultivated and rewarded?

By both sides sharing the risk and sharing the reward, the organization can more easily meet its goals. Projects cannot be completed on time and within budget if both sides are constantly feuding and fighting over risks and rewards. The project schedule and project cost will be held as hostages and bargaining chips in these battles.

Due to the failure of the IT group to perform, many organizations are ready to throw the bums out by outsourcing them. And often the decision to outsource is based on the simple fact that it is more politically expedient down the road (and rightfully so) to throw those external bums out than the internal ones yet again.

The thinking here is that by introducing competition via outsourcing, costs can be driven down. External vendors who specialize can perhaps perform the same task for less money. The reality is that in today's chronically short labor pool, many of the old IT workers now work for the new vendor. These IT workers have new paychecks but the same old office chair. While the economic relationship between the IT provider and the customer has changed, the personal day-to-day relationship has not.

I am a firm believer of shared risk, responsibility and reward between IT providers and IT consumers when tackling difficult software projects. This does not mean that the IT group is any less accountable. It means that both sides have something at stake and have the same rewards. Both sides then have the same goal.

Customer Goals = Organization Goals

In organizations that have internal IT groups, the IT group usually share the same goals as their users. For a manufacturing company trying to remain competitive, operational efficiency might be a goal for the IT group's users. Most of the time, the IT group is aware of this and tries to help the user meet their goals. For external IT organizations, because the two organizations are separate entities, goals do not have to be so neatly aligned. The manufacturing company might work with an IT contractor to build a system that increases efficiency, but the contractor does not usually get rewarded for the increased efficiency. The contractor gets rewarded by getting paid, which usually has nothing to do with increasing the companies manufacturing efficiency.

Whether you are an internal or external IT organization, you must constantly think and re-think your goals. Are they in line with your customers? If not, should they be your customers? Or should you change your goals?

Deepen Relationships and Reduce Costs

All IT providers must find ways of deepening relationships with their users. This helps align interests and goals, helps both sides understand and share risks and rewards and increases understanding of each other. Ultimately, all of this can increase efficiency.

For example, the typical testing cycle in many organizations goes like this:

1. IT provider builds and tests software
2. IT provider delivers software to customer
3. Customer tests software, repeating tests in step 1

In a shared risk, shared reward system, the steps go like this:

1. IT provider builds and tests software

2. IT provider delivers software, test plans and test results to customer

3. Customer tests software but does not repeat tests in step 1 based on test results given in step 2

In this scenario, IT provider and customer share intimate secrets (test plans and test results) and can thereby achieve efficiencies in testing. They can shave time off the schedule and save costs and increase software quality by better breadth of testing. Amazing, isn't it?

In order for this to happen, both sides must *trust each other* and not try to punish each other for exchanging secrets. Both sides need to look out for each other.

Repeatability

Great teams do great things regularly not just because they have memories. It's because they have pencils. Albert Einstein put it nicely:

> "A short pencil is more powerful than a long memory."

Record and document

The power of the written word goes unnoticed by IT. Perhaps its because the complexity of many systems is difficult to reduce to language. But great teams find ways of recording and documenting all that they do so that on the next project they can simply read rather than try to jog everyone's memory.

In fact, many methodologies do not do much more than force organizations to document everything, as if documenting construction processes is the most important part of building quality software on time and within budget. It isn't. While documentation is an important part that should never be ignored, it is just a small part of the picture.

While many software projects share some things in common, no two are alike. A battle plan that was successful on one may not be successful on another for a variety of technical but mostly political and cultural reasons. I don't think of documenting the various pieces of the software process as creating a cookie cutter that stamps out identical copies but as a musical score that can be improvised upon to meet the needs of the moment. Without that score, it is difficult to make any music. But with it, some clever and pleasing improvisations are possible.

Mentor and Train

Documenting is one thing. Getting what's documented into the mind of another developer is another. This is where mentoring and training is important. Here's a very typical day in the life of the new person on the project:

> "I arrived at my desk on my first day. My manager gave me the system documentation and said, start reading. The next day my manager showed me the source code configuration and handed me the change request and asked me to start. Over the past few weeks, no one has had the time to actually explain things to me in more detail."

Mentoring and training is a very time consuming process that even for small projects may take days to conduct fully. Accept this fact and embrace it. Team managers will quickly realize the cost of turnover is

in training and mentoring. Since turnover will occur, make training and mentoring a regular activity. This will have the following advantages:

1. It will make more IT team players skilled in more applications, thus providing management with a bigger pool of backup talent.

2. It will broaden IT team players skills.

3. It may spare management from having to perform a last minute person-to-person mind meld when someone decides to leave unexpectedly.

4. It may allow a team player to break out of an old role into a new one.

5. It will prevent a team player from holding the application hostage until certain demands (like free lunches and pizza) are met.

Great teams are great at teaching new team members.

Build Knowledge Repositories

Great teams often take the concept of documenting and recording things to the next level. They will actually build centralized repositories of this information that is easily accessible and searchable by others. The repository becomes a data well where other thirsty team players come to drink. In this manner, knowledge is now shared among a wider audience thereby allowing the IT grow at a faster rate than it could by using the oral tradition to convey knowledge.

Reward Everyone For All of This

Trivial as it sounds, it is actually easy to make all this recording, mentoring and knowledge repository stuff to happen. Just give cash bonuses (or any meaningful reward) to team players who actually do this. Give bonuses to not just producers of this information, but consumers as well. After all, Discover Card does not give cash-back bonus awards to companies who provide products and services but to the consumer of the service for a very good reason. It is the act of consumption of a service that makes it valuable, not it's production.

Encourage consumption of project improvement ideas, not just production.

Conclusion

If you haven't already noticed, I have not discussed methodologies too much. I've concentrated on less tangible and less measurable things. But consider the following question list:

1. Does your team practice?
2. Does your team practice intensely?
3. Is your team passionate about project details?
4. Is your team passionate about administrative details?
5. Do you make expert use of role players?
6. Do you seek to reduce pain and avoidance where appropriate?
7. Is your team consistent?
8. Is your team persistent?

9. Does your team like to kick it up a notch?
10. Do you cleverly plan small skirmishes?
11. Are you careful about revolutions and death marches?
12. Does your team actively seek replacements for themselves?
13. Does your team never criticize the customer?
14. Do you fight for the customer's best interests?
15. Do you seek to find and measure the bad news as early as possible?
16. Do you actively and continually seek ways to improve productivity?
17. Do you measure and prioritize the irrational?
18. Do you actively seek to align employee and organization goals and interests?
19. Do you actively seek to align customer and organization goals and interests?
20. Do you record and document numerous details about each project?
21. Do you actively and continually mentor and train?
22. Do you positively reward everyone for absolutely all of the above?

If you answered a resounding yes to all the questions above, would you need a total project methodology? Maybe not. Would it still be nice to have one? Probably. Would the methodology be the sole reason you are capable of producing great software on time and within budget. Of course not.

Methodologies cannot address the problems regarding building quality software on time and within budget unless the culture, habits and instincts of the team and its members are changed. Policies, procedures, project plans, development methodologies and all of this ilk are doomed to be useless shelfware until IT managers realize that what really separates great teams from good teams is the discipline and passion to answer yes to all of these questions.

This is not meant to denigrate methodologies. I am a firm and ardent believer in them. But in order to meet the demands of increased software complexity and sophistication, increased technology changes and increased pressures from our users, we need to address these intangible issues along with and perhaps with more force than the mechanics of methodologies.