Dramatically Increasing Project Success with Radical QA:

A Guide to Managing Expectations and Results

Overview

After 50 years of computing, system development projects are still struggling to succeed in being on-time, on-budget and on-quality while delivering all of the functionality that the customer wants and needs. The Standish Group reported in 2003 that only 34% of projects succeed, while 27% experienced cost overruns of over 20%, and 15% of projects failed altogether. On average projects overran their original schedules by 82%, but delivered only 52% of required features and functionality. In real dollars, $55 billion was wasted in 2002 on failed projects and cost overruns alone, while the time overruns and unrealized functionality represented many more billions of lost opportunity and lost productivity.

> "Detailed commitments improve the chances of meeting each others expectations about the process and result of a project."

The reasons for failures, overruns and under performance of projects are well publicized and widely understood, as well as the solutions and prescriptions to promote successful projects. Why then is it so hard to develop systems in keeping with the estimated cost, schedule, quality and functionality? Are the expectations set too high, and are the estimates unrealistic given past experience? Or do customers and developers put those estimates beyond reach by their conduct on a project? Of course, it's both. But what can customers and developers do differently to dramatically increase the chances of success for their project? We say the answer is Radical QA[1].[2]

Before any contract is signed, Radical QA ensures that the commitments made by the customer and developer about the cost, schedule, quality and functionality of a project are realistic and detailed. Realistic commitments are attainable and sustainable. Detailed commitments improve the chances that the parties will meet each others expectations about how to conduct the project and how to stay on time, on budget, on quality and on functionality with a minimum of dispute.

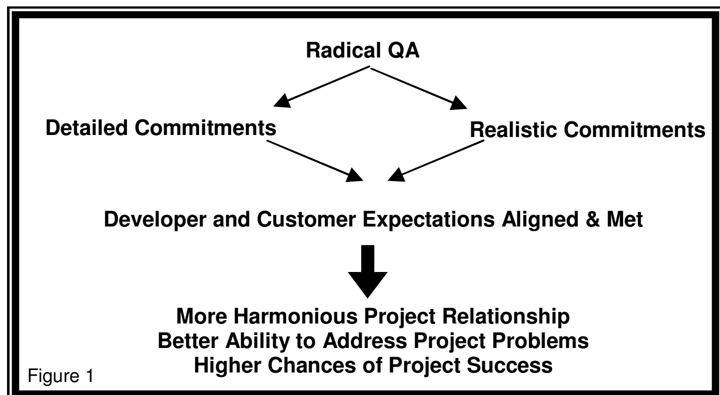

Figure 1

Radical QA is not radical because it uncovers new problems or delivers new solutions. It is radical because it addresses well known problems with well-known solutions *before* rather than *after* a contract is signed. By enshrining those solutions in the project bargain before development work on the project begins, Radical QA gives projects a better chance of succeeding.

[1] Pending Service Mark, Warren S. Reid, Lubomyr Chabursky, 2004

[2] This article was written with the assistance of observations, comments and suggestions from Warren Reid, Managing Director WSR Consulting Group, LLC.

© Copyright 2004, Lubomyr Chabursky, All Rights Reserved

Optimism, Overstated Promises and Lack of Caution

The whole point of a contract is to put onto paper a bargain struck between two parties. In systems development the bargain is typically about what a developer is willing to deliver by an agreed date in return for specific payment by the customer. But the whos, whats, whens, hows, and ifs of that bargain can be voluminous, complex and intricate. Each party goes into a project with its own expectations about the details of who will do what, when, and how, and a contract should be able to put the expectations of both parties onto the same page, literally and figuratively.

> "A contract should be able to put the expectations of both parties onto the same page, literally and figuratively."

The greatest threat to unifying the expectations of the developer and customer is the optimism and euphoria that encompass the courtship and honeymoon period of a project – the announcement of the project, the ensuing bidding competition, and the contracting phase. During that period everyone is excited about the new opportunity. Developers present themselves in the best light, which does not include the yet-to-be explored ifs, ands or buts inherent in most projects. For their part, in their enthusiasm to do something special and unique for their organization, customer managers are particularly receptive to the optimistic and lofty promises made by some developers about their abilities and proposed solutions.

With little opportunity to delve into the customer's stated requirements, developers prepare elaborate presentations that take those requirements at face value and amplify the benefits of their solution. Instead of critically assessing the reliability of bidders' representations, customer managers often exploit them to bolster the project's business case, and improve the chances of getting management and executive approval for the project. Both parties feel an urge to sign a contract and get on with the project as soon as possible – if only the lawyers could speed up their crossing of the "t"s and dotting of the "i"s.

Diverging Expectations

The problem with the excitement and urge to proceed is that it does not allow enough time for the customer and developer to explore the user requirements more fully, to expose assumptions, to consider different options and choices of functionality, to appreciate some of the complexities of development, and to revise the requirements in light of a better insight into the impacts on the customer's business processes. If the developer's representations were based on an abridged understanding of the customer's requirements, user preferences or business environment, then the developer's representations about the effort of development and delivery of functionality may prove to be unachievable and of little use.

> "Most contracts do not reflect a full and mature understanding of what needs to be built and how it needs to be built."

Similarly, the rush to get on with a project leaves less time to flesh out commitments about how to implement project practices during the development. Hence, most contracts are crafted and signed based on effort, cost, schedule, quality and functionality commitments that do not reflect a full and mature understanding of what needs to be built and how it needs to be built. Too often, the full and mature understanding of the development is left to take shape after the contract is signed and once the work has begun.

© Copyright 2004, Lubomyr Chabursky, All Rights Reserved

There is a natural tendency for the expectations of both parties to drift apart because the customer and developer fight for the same finite pie of project resources. More compensation and more time for a developer to undertake a given task, means higher costs and longer delays for the customer. Back-slapping "win-win" pronouncements made during the honeymoon period are replaced by the careful scrutiny of contractual commitments to get the most of the resource pie. That in turn propels expectations of the parties to evolve in opposite directions.

Such diverging expectations about project practices, functionality, schedule and cost guarantee that both parties will fail to meet each other's expectations. When that happens, each side starts to begrudge the other for not living up to the bargain. As the relationship and goodwill between the parties suffer, the give-and-take necessary to address unanticipated issues and problems becomes more difficult and tortured. In some cases, the customer-developer relationship turns dysfunctional and the entire project fails.

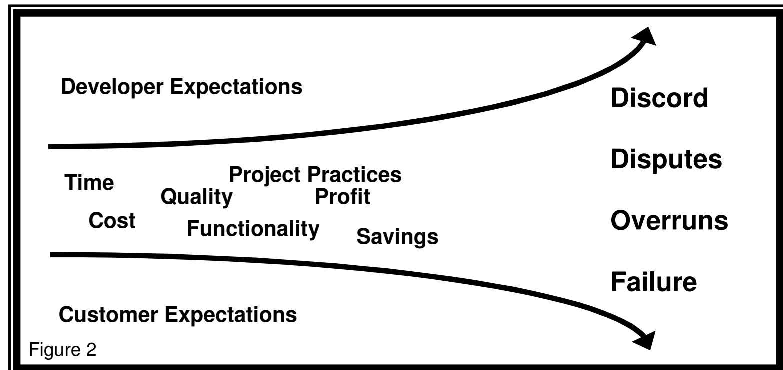

Figure 2

Radical QA

Radical QA is a method of putting the parties on the same page at the outset of the project, and keeping their expectations synchronized going forward. With Radical QA the commitments of the parties regarding cost, schedule, quality, functionality, and project practices are scrutinized before the contract is signed by someone capable of maintaining an objective perspective.

> "Radical QA is radical because it addresses well known problems with well known solutions *before* rather than *after* a contract is signed."

Just as testing relies on personnel independent of the programming personnel to try to break the code and find the errors, omissions and defects, so too does Radical QA rely on someone who is not caught up in the euphoria and excitement of the pre-contractual honeymoon period to verify that the parties are on the same page in terms of their expectations, and to determine that those commitments are detailed enough to keep the parties on the same page.

Radical QA is radical because it applies QA processes to a phase of the project that had hitherto been free of scrutiny, namely, the honeymoon period during which the parties formulate the commitments that will govern their relationship and the conduct of the project.

© Copyright 2004, Lubomyr Chabursky, All Rights Reserved

Types of Commitments

There are three categories of commitments:

- "What" commitments
- "How" commitments
- "Business" commitments

> "Euphoria of "win-win" is soon replaced by the careful scrutiny to get the most of the project resource pie."

1. "What" Commitments

The "What" commitments define the 1) time, 2) cost, 3) functionality, and 4) quality of what the project will deliver to the customer. Time commitments are typically expressed in the contract as an overall project duration, delivery deadline, and a schedule of deliverable milestones. Cost commitments are expressed as an overall price, a project budget and payment milestones. Functionality commitments are expressed as a statement of work, user requirements, functional specifications and similar descriptions of the interim deliverables and final product. Quality commitments are expressed as quality assurance metrics, service level commitments, and user acceptance criteria.

2. "How" Commitments

The "How" commitments include the range of project practices and project processes that define how the developer and customer will work together, and their respective roles and responsibilities. They include such things as project communication, staffing, methodology, quality assurance, testing approach, conversion methodology and data accuracy, acceptance process, project management, schedule tracking, risk management, quality management, scope management, configuration management, automated tools, collaborative software, development environment management, project network access and security, conflict escalation and resolution procedures, and documentation. These are the kinds of things that need to be detailed, discussed, and agreed to, so that the parties will know their role in the conduct of the project.

3. "Business" Commitments

The "Business" commitments consist of the parties' understandings about their respective roles, as well as about the profits, savings and collateral benefits that the parties expect to achieve from the project. The following are examples of business commitments: 1) The customer undertakes to take "ownership" of the development process in order to expedite knowledge transfer; 2) The developer agrees to a low contract price, in return for an expectation that the customer will be lenient on scope changes; 3) The developer offers a low price in return for joint marketing of the solution and the developer's services; 4) The parties commit to make decisions quickly and decisively; 5) Due to schedule constraints, the customer is willing to accept a less robust system that meets the target date, but which would then be completed or enhanced after go-live date; and 6) To reduce cost, the developer's systems integration and project management responsibilities are restricted, and their liability is capped at a low level.

© Copyright 2004, Lubomyr Chabursky, All Rights Reserved

Radical QA Rules

Radical QA applies 7 rules:

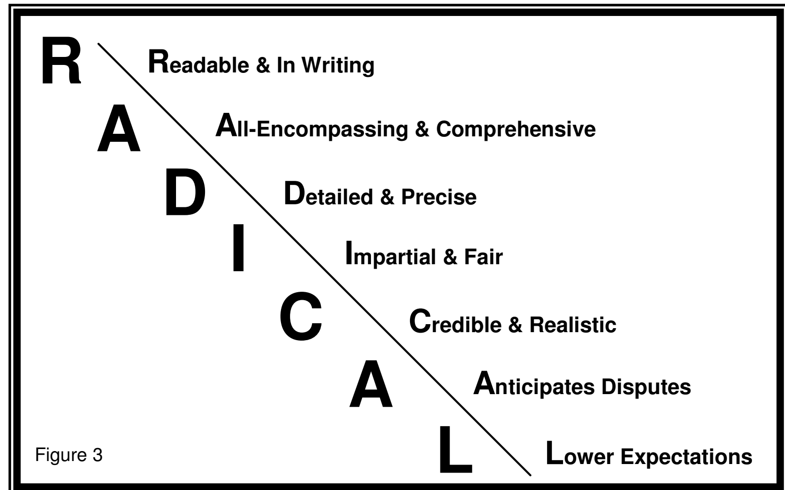

Figure 3

R – Readable

A – All-Encompassing

D – Detailed & Precise

I – Impartial & Fair

C – Credible & Realistic

A – Anticipate Disputes

L – Lower Expectations Where Appropriate

Rule # 1: R → **Readable**
ADICAL

The first rule is that all commitments should be put in writing. As people and roles change, recollection of conversations and verbal commitments change also. Even on the day of the conversation, different people will hear different things, and may come away with their own understandings of what was said or agreed to. The personal memories of those who were "in-the-room" tend to evolve over time, and rarely do all participants remember the same thing. While people can always disagree on what words mean, putting the words into writing provides an objective basis to assess what was said or agreed to, and avoids the problem of changing recollections.

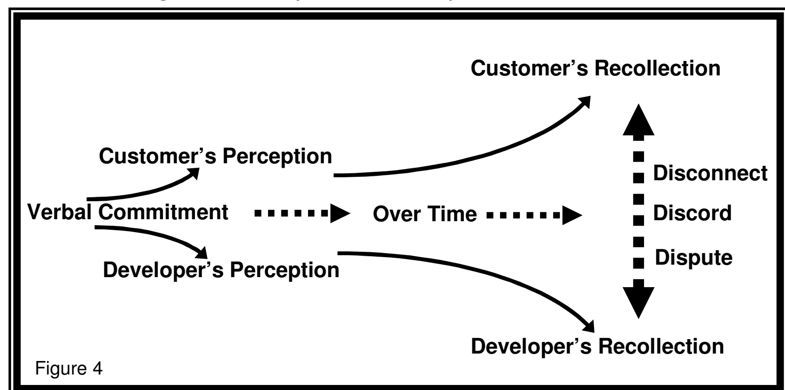

Figure 4

© Copyright 2004, Lubomyr Chabursky, All Rights Reserved

Rule # 2: R(A)→ **All-Encompassing** DICAL

The second rule is that all commitments should be comprehensive. During the bidding process, the customer and developers make all kinds of representations about their needs, abilities, methods and environment. Those representations create expectations on both sides whether or not they are addressed formally in the contract. Unless representations are addressed together by the parties, then one party will form an expectation on its own without the benefit of confirming the accuracy of that expectation with the other side. Moreover, given that both parties are vying for the project's finite pool of time, money and people resources, the expectations arising from representations made are unlikely to match on both sides.

Customers will take for granted that developers will provide special features or functionality that may have been referred to in a sales pitch - whether or not that feature is referred to in the contract. Customers feel betrayed if the developer later cites the "complete agreement" clause that lawyers often add to contracts to argue that a particular special feature was not part of the bargain. This leads to disputes over whether the functionality is in-scope or out-of-scope, and who absorbs the consequences. The ubiquitous complete agreement clauses are totally useless at stopping disputes during a project, but are very good at inspiring creative arguments in litigation after a project has fallen apart.

Customers and developers will even form expectations about issues that were not discussed during the bidding process. For example, customers often expect that when staff turnover occurs developers will replace departing development personnel with equally qualified and experienced personnel, even if the developer made no such representation. Each project has an inherent waterfront of issues that arise from the nature of the project, whether or not the parties chose to talk about each and every one of them during the bidding and contracting process. Any hole in that waterfront of issues left by the failure or omission to discuss a particular issue will be filled-in by both sides with their own expectations.

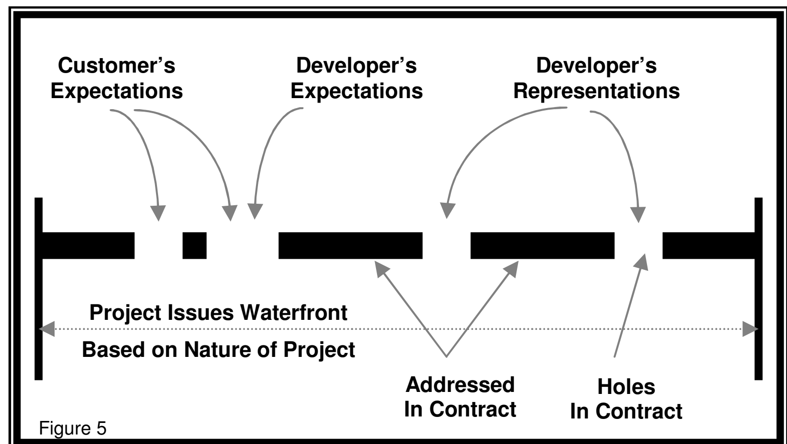

Figure 5

For example, functionality may require choices between different options of varying complexity. A customer may feel entitled to get, with no additional payment, the option that is more difficult to build, even if re-work is required to accommodate the functionality. The customer will argue that the option was implied in the contract, and that the developer overlooked it during development. If the developer argues the opposite, a dispute is born.

© Copyright 2004, Lubomyr Chabursky, All Rights Reserved

> "Any hole in the contract will be filled-in by both sides with their own expectations."

If the developer and customer adopt different expectations regarding specifications of functionality, then system acceptance could become a tortured and extended exercise. Indeed, such differing interpretations could cause the developer to produce a system that is perfectly on-the-mark as far as the specification is concerned, but quite off-the-mark of what the customer wanted or needed.

The objective of this rule is to ensure that all representations made during the bidding competition and all issues that comprise the project's waterfront of issues, though left out of the discussions, are addressed one way or another in the contract so the expectations of both sides are synchronized.

Rule # 3: R A **D** I C A L → **Detailed & Precise**

The third rule is that the commitments should be articulated with clarity and precision to minimize the room for differing interpretations. Terms such as "user-friendly", "easily modifiable", "good audit trail", "strong security", "fast recovery and restart", and "best efforts" are just a few examples of a long list of ambiguous expressions that litter systems development contracts everywhere. Such vague language requires subjective interpretation, which changes from one person to the next, and even varies over time with the same person.

> "Radical QA is an objective assessment of whether contractual commitments are detailed enough to keep the parties on the same page."

These types of expressions arise habitually because the customer and developer have not devoted enough time to discuss and pin-down exactly what they want in terms of the end product, or the project process needed to develop it. The true test of whether the product specification language is sufficiently precise and detailed is that you can write test scripts to conclusively and objectively test the system for each term in a specification, without the need to make any subjective assessment.

Parties often find out that the specification was too vague when they get to the systems acceptance stage and seek to apply different tests to test the same component of specification. If any subjective element or interpretation is required to design a test for the specification, then the customer and developer will likely come up with different tests, which guarantee that they will disagree over the acceptance testing of the system and end up in litigation. A good and hard exercise is for the parties to define together what is meant by success, in terms of stability, reliability, maintainability, performance, the batch processing window, performance during peak usage, protection against malware, recoverability, scalability, re-use of code, testability, reparability, and amount of allowable bugs by severity level, amongst others.

Insufficient detail in the product specification is as harmful as vague language. The desire to get on with the project results in general descriptions of the product, which need to be fleshed-out during the development. Often the customer will not come to terms with and make conclusive decisions about its user requirements until its users are forced to commit to detailed specifications, or until they see the various functionality options and appreciate the impact on their business processes. Likewise, developers often do not fully understand what the customer needs, and cannot accurately

© Copyright 2004, Lubomyr Chabursky, All Rights Reserved

explain all of the available choices and their ramifications on the development and resulting business process, until they are forced to articulate the specifications in complete detail.

Many of the features and functionality that are fleshed-out during the project will require more effort and resources than the developer contemplated in their bid. The question then becomes, are those details a clarification of the specifications, and as such should they be treated as in-scope, or are those details the addition of new scope. The answer to that question determines whether the customer will pay more money, and allow more time for the developer to provide that functionality, or whether the developer will have to dip into its own resources and financial margins to provide that functionality.

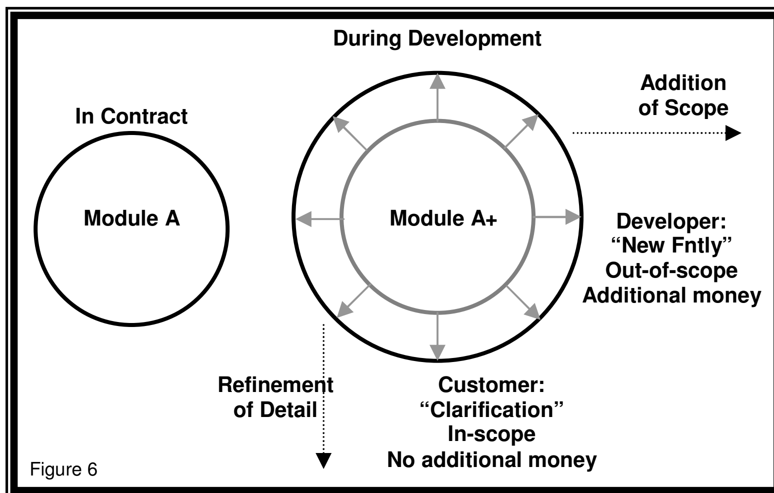

Figure 6

With reasonable parties on both sides, the small stuff will be agreed to through a balanced give-and-take between customer and developer. But all too often, scope disputes erupt when the number of "gives" start to outweigh the "takes" for one or both parties, or when the effort associated with several fleshed-out features of functionality become too large for the normal give-and-take.

Some customers and developers fear that too much detail in the product specification will constrain the creativity of the developer and the flexibility of the parties. The truth is that most parties favor less detail precisely because they do not want to be nailed down, and prefer to preserve the option of arguing a different bargain down the road. Others don't prescribe the details simply because they find it too hard and too time consuming.

> "Most parties prefer less detail precisely to preserve the option of arguing for a different bargain down the road."

No amount of detail restricts any party from agreeing to change the requirements and specifications for something better in the future. But precisely defined functionality helps the parties recognize that a change is being made, and minimizes the waste of time, energy, cost, and goodwill on arguing about whether a change is in or out of scope. The less detail the contract specifies, the greater the latitude for both sides to legitimately argue their own points of view – often to the point of impasse when the number of such disputes render the stakes of losing the argument too high to let go.

Projects following agile rather than planned methodologies purposefully leave the specification of the functionality nebulous, to allow the interaction between the developer and customer to flesh-out the specifics during the development process. Even so, parties using an agile methodology need to agree and define with precision what functionality or how much development effort is covered by the compensation scheme that they agree to.

Precision is equally important for the contractual specification of project practices. While some parties are willing to put their trust in expressions such as, "consistent with industry standards",

© Copyright 2004, Lubomyr Chabursky, All Rights Reserved

"competent and workmanlike manner", or "best practices", these terms can take on different meanings dependent on the particular circumstances of a project, and based upon differing cost, time, and quality restrictions, and the risks that each party is willing to bear. The meanings and expectations of these terms can also vary with differing relative and professed experience of each party.

> **"Precisely defined functionality minimizes the waste of energy and goodwill arguing about whether a change is in or out of scope."**

While diverging expectations about project practices rarely start an overt dispute between the parties, they contribute to the grudges that each party harbors about the failings of the other side, and when a dispute does emerge on a scope issue, these grudges are drawn in to bolster the assertion that the other side is at fault for delay and spiraling costs. Recording in the contract a specific list of measures for each project practice, forces both parties to consider and agree on precisely what steps will be taken, and what steps will be deliberately left out, so that there will be little room for grudges down the road. For example, the parties could and should agree on the maximum duration of a task; how far in advance must all tasks be assigned to specific team members; how many task dependencies can be left unspecified; how often to calculate a critical path using a computer assisted scheduling tool; and the content and format of status reports, as well as their frequency and distribution.

Referring again to agile methodologies, these invoke much fewer project practices and in a much less formal fashion than planned methodologies. It is important that the customer understand that to be the case by so specifying in the contract, and setting out precisely what will be done, and what will not be done.

Rule # 4: RADICAL → **Impartial & Fair**

The fourth rule is that the commitments should balance the interests of the parties fairly. A project has a better chance of avoiding acrimony if managers on both sides look good to their respective constituencies. Customer managers are evaluated on how well they cut costs and manage the project within or below budget. Developer managers are scored on maximizing profit from a project. This is a classic zero-sum game between diametrically opposite objectives: saving costs and earning profits. The managers of both sides will work well with each other if they both look good.

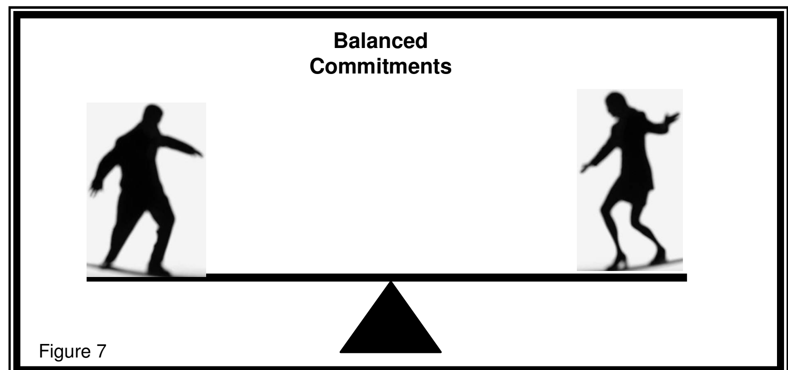

Figure 7

> **"Managers of the disadvantaged party will blame the other side for not living up to their bargain."**

If the interests of one side start off disadvantaged or if they become so, then it is likely that managers on that side will work hard to shift the blame to the other side for not living up to their bargain or acting in a high-handed manner. In other words, achieving for

© Copyright 2004, Lubomyr Chabursky, All Rights Reserved

oneself a more favorable position in the contract is unsustainable and counterproductive, because the losing side will do its utmost to realign the bargain during the project or afterwards during the warranty and maintenance periods.

While it sounds counter-intuitive to strike a bargain and manage a project to make your counterpart look good, it makes eminent sense in projects, given that success is maximized through cooperation and goodwill.

Rule # 5: RADI**C**AL → **Credible & Realistic**

The fifth rule is that the commitments should be credible and realistic in terms of past experience for projects of the same type, nature and size. The commitments regarding the schedule, cost, quality and functionality of a project represent the customer's and developer's expectations. If those commitments and expectations are unrealistic, then both parties will blame the other for falling short.

The euphoria and optimism that pervades the honeymoon period leads customers and developers to put the factors governing time, cost and effort into their best light. The rush to proceed with the project prevents developers and customers from becoming fully informed about user requirements and functionality before estimating time, cost and effort. Sometimes, developers simply accept an end date and budget that the customer sets arbitrarily as a lesser evil, rather than risk losing the bidding competition through caution and frankness.

Past experience has been quantified in a number of databases and metric methodologies such as, function point counts, COCOMO I and II, Capers Jones metrics, amongst others. Occasionally you'll find managers who actually believe that they can beat the odds and run a project better than past experience suggests for projects of the same type, nature and size. While some projects actually perform better than past experience, prudence does not plan on such pleasant surprises.

When using estimation methodologies, it is crucial that the factors and business drivers be properly, accurately and honestly set up – otherwise the estimates based on historical averages could vary by as much as 50% to 500% from what they should be. The fact that most developers come up with estimates that are 50% of average estimates for past projects, demonstrates that as a profession, developers and programmers are overly optimistic in nature – they always believe that they can do it better, faster and cheaper than anyone else on the planet.

Even the best estimates do not take account of scope creep. Scope creep is not a dirty word. It is a reality that is a natural part of any buying process. No one buys an appliance or car limited strictly to the price and features that they had in mind before embarking on a shopping expedition. What a customer wants will inevitably change during the buying process, as the customer gets more familiar with the product through interaction with its features, because that's how humans learn. Similarly, as users get closer to the system functionality and better understand the impact on the business processes, the customer, together with the developer, will discover new possibilities and improvements – many of which will be "must-haves". In fact, scope creep is the only way that the customer really gets what its users want and need.

> "Scope creep is the only way that the customer really gets what its users want and need."

© Copyright 2004, Lubomyr Chabursky, All Rights Reserved

Developers should prepare customers to expect scope creep. More importantly, developers need comprehensive and detailed product specifications to identify the expansion of system scope. Likewise, it is important to create detailed project plans to identify and calculate the precise amount of cost and schedule impact associated with each increase in scope, so that such increases could be justified by the increased benefits for the customer, rather than be tarnished with the same brush as all other cost and budgetary overruns that occur for different reasons.

Rule # 6: RADIC**A**L → **Anticipates Disputes**

The sixth rule is that commitments should anticipate disputes. Even the best worded contracts cannot prevent misunderstandings and differences of opinion that could lead to disputes. Major categories of commitments, such as schedule, cost, functionality, quality, and project practices should build-in metrics and mechanisms to resolve disputes that will arise. For example, the contract should specify a process on how to resolve a scope dispute, and should specify a process to absorb delay. The more consequences and resolutions that can be agreed to in advance, the easier it will be to move beyond disputes when they occur during the project.

> "As lost expectations erode goodwill, problem-solving becomes more difficult."

As a general rule, disputes must not be allowed to aggregate and fester on the sidelines. Parties should agree to a real-time arbitration process, such as through a Special Project Adjudication Team, whose role is to resolve all disagreements when they are spats, rather than when they aggregate to become disputes.

Rule # 7: RADICA**L** → **Lowers Expectations Where Appropriate**

The seventh rule is that expectations regarding commitments should be lowered wherever appropriate. It is far better for the parties to be surprised by over-performing rather than under-performing on commitments.

Timing

Radical QA calls for a degree of open, frank and honest discussion between the developer and customer that most parties are not accustomed to. During the period leading up to the contract, developers focus on winning the business and getting the most favorable terms. They avoid saying anything that would put off the customer or lose the edge in a competitive environment, or later hurt their negotiation posture. What good is all that honest talk when you end up the loser in a competition? That is a legitimate concern for anyone who has a business bottom-line to meet, and does not have the luxury of enjoying an ivory tower. From a developer's perspective the optimal time for Radical QA should be once the competition is over, but before the contract is finalized.

© Copyright 2004, Lubomyr Chabursky, All Rights Reserved

However, a well-informed customer who embraces the advantages of Radical QA should invoke the process and encourage its use from the very start, including the formulation of the competition, and interactions with the bidding competitors.

That approach may introduce an additional delay that the customer had not likely counted on, as the developer and customer work out the What, How and Business commitments in accordance with the seven rules of Radical QA. When faced with a customer who is not willing to wait for that process to create sound and realistic commitments, each developer must decide whether they want the business badly enough to weather the possible acrimony and discord that could slow down and perhaps even sink the project.

Applying Radical QA at the outset to synchronize the parties' expectations is only the beginning. Unforeseen events will take place during the project, and changes in personalities on either side will affect the values and priorities of the parties. Thus it is inevitable that expectations themselves will evolve and change over time. Ongoing Radical QA can keep both sides aware of how those expectations change, and allow them to evolve their expectations in tandem.

© Copyright 2004, Lubomyr Chabursky, All Rights Reserved