by Robin F, Goldsmith, JD
robin@gopromanagement.com www.gopromanagement.com
This article will appear in ASQ Software Division's *Software Quality Professional* magazine

Abstract
Contrary to common perceptions, "software process improvement" is not synonymous with or limited to well-known branded broad-based formal initiatives which tend to involve considerable overhead expense and delay before benefits materialize, if indeed they ever do. Alternative low-overhead techniques offer considerable advantages because they are more likely to be implemented, generally provide benefits sooner and cheaper, and frequently are more pertinent to an organization's specific needs. Seven low-overhead improvement techniques are described, including: generic REAL process improvement, skills enhancements, management good practices, and four engineering good practices—REAL requirements, reviews, reuse, and Proactive Testing™.

Keywords
REAL Process, Presumed Process, REAL business requirements, Proactive Testing™, software process improvement, metrics, management practices, reuse, reviews.

Introduction

The term "software process improvement" often brings to mind high-overhead formal commercially-branded initiatives. Those seeking improvement may not even be aware that low-overhead software process improvement techniques not only are available but may be a faster, cheaper, and in some ways better alternative. Organizations frequently can gain considerable advantage from simpler alternatives which actually are implemented and are providing benefits, even if they are smaller, as contrasted with waiting much longer for potentially larger benefits which may never actually materialize or ultimately may not justify their cost. Moreover, low-overhead methods ironically sometimes can provide superior results by overcoming often-unrecognized weaknesses in their larger and more complex cousins.

High-overhead approaches characteristically involve expensive, extensive, extra top-down administrative organizational structures to guide the initiative and significant up-front broad-based training of most employees before benefits begin. Ordinarily, the approaches also mandate a number of formal procedures and paperwork be added to regular workflow. Executive support is essential; and conformance often is enforced through command and coercion, frequently resembling evangelical religious fanaticism. For instance, one prominent organization was widely-known for summarily firing not just those managers who privately questioned their branded program, but also those who merely failed to publicly demonstrate adequate active advocacy for it.

Because of the high start-up costs and relatively long delay until benefits begin accruing, or perhaps because of skepticism about whether the programs actually will produce needed benefits, many organizations choose not to undertake these often massive efforts. Moreover, some organizations abandon their high-overhead initiatives after making significant investments, for as even lead Capability Maturity Model (CMM®[1]) author Mark Paulk said, "Many—perhaps most—software process improvement efforts fail."[2]

by Robin F, Goldsmith, JD
robin@gopromanagement.com www.gopromanagement.com
This article will appear in ASQ Software Division's *Software Quality Professional* magazine

1. Generic REAL Process Improvement

Many of the big branded approaches include often sizeable and elaborate models of presumably ideal total software processes. Merely becoming familiar with such massive material can be a formidable task, which can pale in comparison to the difficult effort imposing the model processes in one's own organization. Moreover, such overwhelming shotgun-like blanket changes can amount to overkill, possibly replacing existing processes which work fine, perhaps with frankly less appropriate ones, and implementing superfluous processes for things the organization doesn't really need.

Rather than imposing some presumed ideal process, generic process improvement analyzes the current processes and focuses rifle-like improvement efforts only on those with opportunities for significant benefit. Ironically, generic process improvement is somewhat of a lost art, largely because the branded approaches have virtually usurped the terms "process" and "software process improvement," though not necessarily with conscious intent. These days, many people think "process" means massive formal procedures and mandated documentation-heavy busywork, typically associated with a particular branded model process; and "software process improvement" for many has become synonymous with imposing a branded model's processes on one's organization.

REAL vs. Presumed Process

Although often not explicitly recognized, even in much of the literature, implicit in generic process improvement is the need to identify and improve the REAL Process. *Many (and probably most) generic and branded process improvement efforts fail because they don't properly understand what a process is and therefore don't identify the REAL process that needs improvement*.

Many attendees in my seminars/speeches use expressions like, "How much process do I need?" Most define a "process" as "a set of steps one is supposed to follow to turn inputs into a particular result" or words to that effect. That's the definition of a *defined process*, and as typically interpreted, it generally refers to a defined process mandating often extensive formal procedures and paperwork.

Realizing that such common perceptions actually demonstrate misunderstandings about what processes really are helps explain why so many process improvement efforts fail to deliver expected results. To be effective, improvement efforts must start with this more accurate definition:

> ***A process is a set of actions, beliefs, practices, and customs that taken together produce a result.***
> A result is the inevitable outcome of the process followed, regardless whether that process was intended, desired, defined, or even recognized.
> Repeating the process produces essentially the same result.

by Robin F, Goldsmith, JD
robin@gopromanagement.com www.gopromanagement.com
This article will appear in ASQ Software Division's *Software Quality Professional* magazine

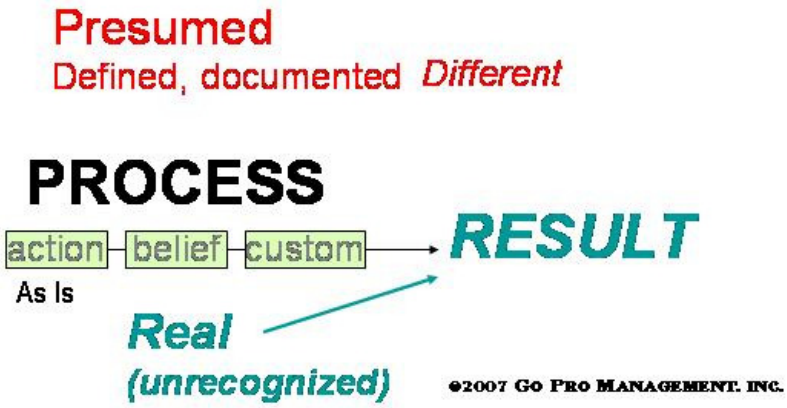

Figure 1. REAL vs. Presumed Process

As can be seen in Figure 1, there are actually two processes. The *REAL Process* is the As Is process that actually is producing the current results. The REAL Process includes beliefs and customs, not just procedures. Very often, people are not aware of the REAL Process. The *Presumed Process* is what people think is producing the results. Often the Presumed Process is defined, which means that people who are privy to the definition would describe the process in basically the same way. A process can be defined without being documented, although often the process definition is in the form of written documentation. Many defined processes have no formal procedures, or at least none involving paperwork. An example of a defined process which is not written down occurs when someone holds a door open for you and you say, "Thank you." Everyone knows to do it.

Very often, the Presumed Process is different from the REAL Process, which almost always happens when the REAL Process is not recognized. For example, the 55 miles per hour speed limit on a highway is a defined and documented Presumed Process; but the REAL Process is the 70 plus miles per hour that everyone is traveling. Highway engineers know they must design the highway to be safe at 70 miles per hour and higher, because a highway designed with the expectation that drivers actually go only 55 miles per hour would result in carnage.

by Robin F, Goldsmith, JD
robin@gopromanagement.com www.gopromanagement.com
This article will appear in ASQ Software Division's *Software Quality Professional* magazine

As Figure 2 shows, when we want to change our results, we must change our process from the As Is process producing our current results to a Should Be process that will produce the desired changed results.

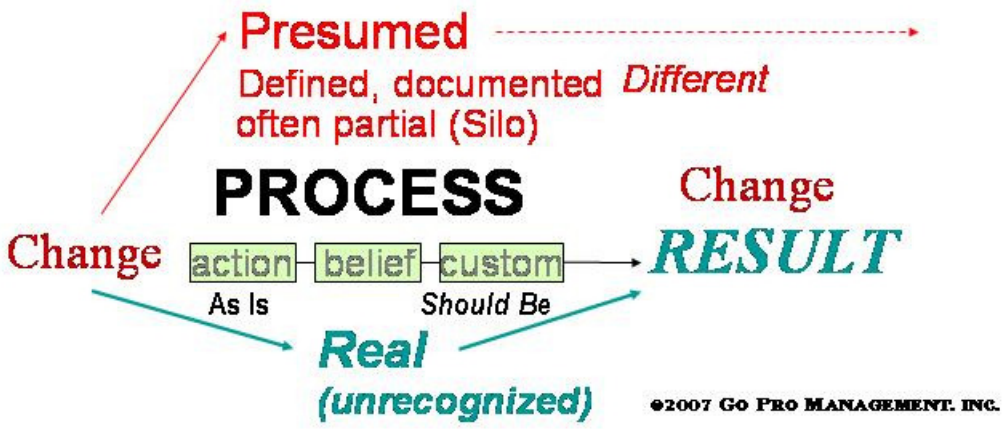

Figure 2. Changing REAL vs. Presumed Process

But, which will people change, the REAL Process they may not even recognize or the Presumed Process they think is producing the results? Obviously, they'll change the Presumed Process; and if the Presumed Process is different from the REAL Process and is not producing the current results, changing the Presumed Process will not have the desired effect on the results. In fact, changing a Presumed Process which is not the REAL Process producing the current results can be counterproductive, wastefully diverting valuable time and resources. Therefore, to change one's results effectively, one must change the REAL Process producing those results; and before one can change the REAL Process, one must recognize it. That may not be easy.

For example, consider the scenario described in Figure 3. I'm sure you are familiar with projects where coding and testing are planned for particular amounts of time which end at the Stop sign deadline; and then the coding takes longer than expected. If the amount of necessary testing is proportional to the amount of coding, and the amount of coding increases, then the necessary amount of testing also should increase. But what doesn't change? Usually the deadline stays the same. Testing gets delayed, squeezed, and cut short of the originally planned amount, not to mention being way short of the revised amount; and the project reliably and predictably delivers trouble.

by Robin F, Goldsmith, JD
robin@gopromanagement.com www.gopromanagement.com
This article will appear in ASQ Software Division's *Software Quality Professional* magazine

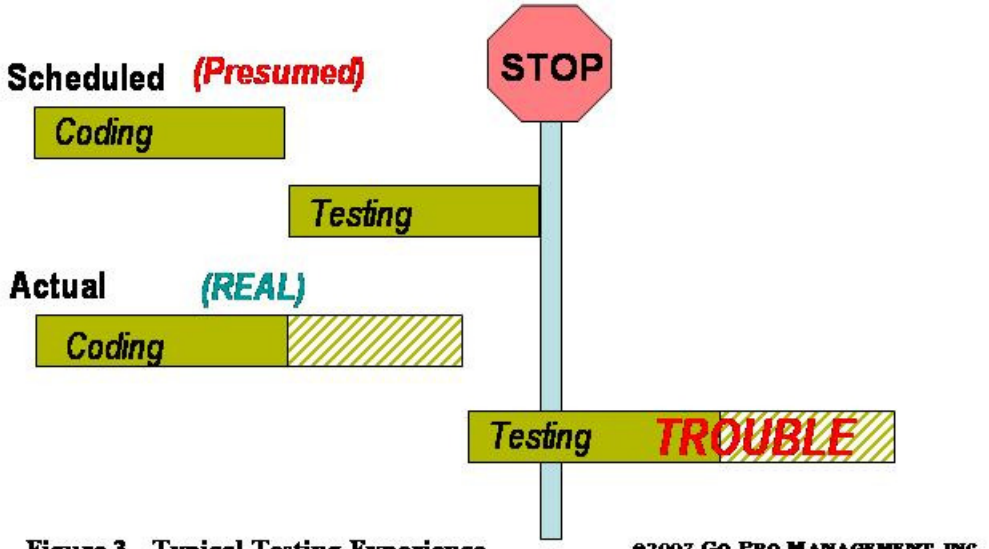

Figure 3. Typical Testing Experience

Have you ever seen this happen? Everyone in Information Technology (IT) has. Have you seen it happen more than once? Undoubtedly so. Would it be fair to say it happens practically every project? Most IT folks I meet say, "Yes." Well, if this is what happens every project, then your REAL software process is to plan coding and testing, blow your planned schedule, squeeze your testing, and deliver trouble.

Yet clearly nobody in your organization thinks, let alone would say, that's your software process. Instead, they'd say the organization's software process is the Presumed Process: schedule time for coding, followed by time for testing, followed by delivery on the deadline. When they try to improve, they'll make changes to the Presumed Process, which doesn't really happen, and won't be likely to change the REAL Process's repeatedly predictable production of trouble.

Look around your organization with newly opened eyes and you'll see countless examples of unrecognized REAL Processes that differ markedly from Presumed Processes. Perhaps the biggest Presumed Process is that the IT organization is good at developing systems, when in fact the REAL Process is that systems are routinely late, over-budget, and not what stakeholders need. You should also start understanding why your REAL Process probably includes repeatedly not being able to cost-effectively make appreciable improvements despite making large efforts and possibly enlisting well-known high-overhead branded software process improvement approaches.

by Robin F, Goldsmith, JD
robin@gopromanagement.com www.gopromanagement.com
This article will appear in ASQ Software Division's *Software Quality Professional* magazine

Silos

There's a good chance that part of your REAL Process difficulties are attributable to silos. Frequently, the presence of silos makes it especially hard to see the REAL Process. Silos occur when parts of an organization, such as a particular department or function, view only a narrow slice of the REAL Process without being aware of the full process, such as other departments and functions. Viewing a slice of the REAL Process out of context effectively creates a Presumed Process view which is different from the full REAL Process. Thus, to get meaningful context, the REAL Process by definition must be viewed in its entirety, from beginning to full end result.

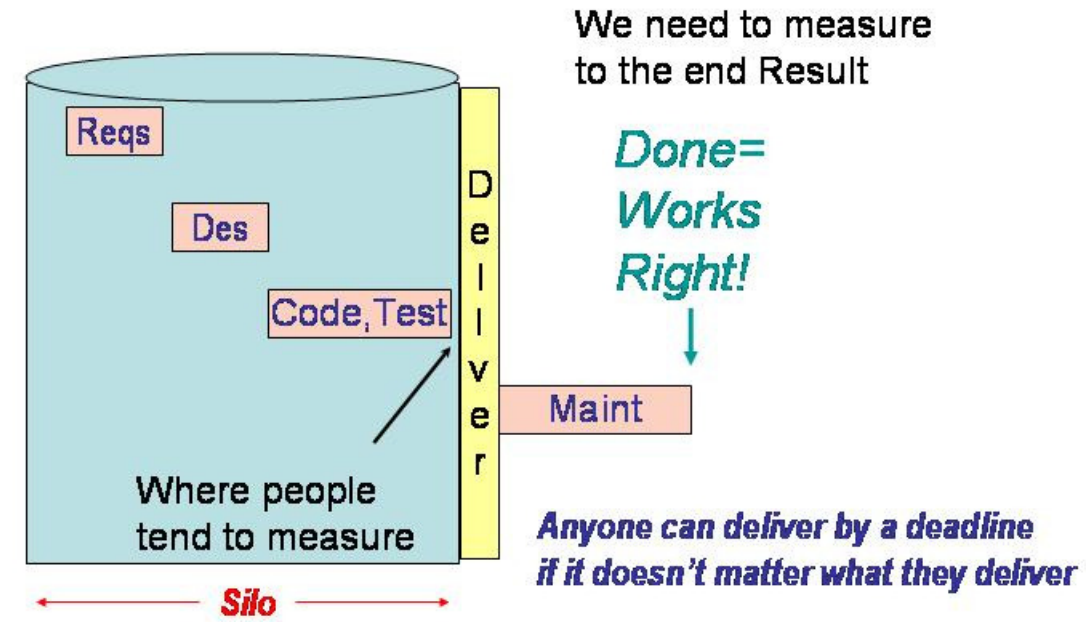

Figure 4. Typical Development Silo

Figure 4 describes the way most IT organizations manage development projects. The project is presumed to end at the point of delivery, and most organizations tend to have a single measurement data point—the deadline.

If yours is like most organizations, this is pretty much the way things are done, project after project, which makes it your REAL Process. Dressing it up with lots of paperwork and ceremony doesn't change the underlying REAL Process, or its predictable problems.

When is the project really done? It's not when the deadline is reached, because anyone can deliver by a deadline if it doesn't matter what they deliver. Rather, the project is

really done when the project deliverable *works right*; and everybody in IT knows that almost always additional post-delivery effort is needed to make what was delivered at the deadline work right. You may have a more prosaic euphemism for these finishing touches, but basically they're maintenance.

Your organization also probably accounts separately for the "project" through delivery and for post-delivery activities, but doesn't tie the fix-its back to development practices that necessitated the extra work. That's a defined procedure which institutionalizes a Process that erroneously Presumes pre- and post-delivery activities are unrelated and thereby virtually assures the full REAL Process will stay unrecognized and perpetuated.

Practically every endeavor except IT, and certainly any which is recognized for its effectiveness, knows (1) that meaningful measurement requires more than a single data point and (2) that the consequences of behavior need to be included in the costs of that behavior. For instance, warranty expenses are charged to the manufacturing process. Making the IT delivered product work right is like warranty repairs. The REAL software development process needs to include post-delivery maintenance repairs needed to make the software work right.

If You Don't Know What You're Doing, You Don't Know What You're Doing

You need to know what your full REAL Process activities are, how much you're expending on them, and what you're getting for your time and effort. If you don't have appropriate quantified measures of your results and what's REALLY causing them, you literally and figuratively don't know what you're doing.

A football coach would be fired instantly if he didn't know the score of the game, down, distance for a first down, and number of yards gained and lost for each play/formation. Yet, I'd contend ignorance of comparable essential measures is the REAL Process in most IT development organizations. No wonder football coaches get paid in the millions and IT jobs get sent overseas to the lowest bidder.

Be careful of counting on high-overhead imposed processes to cover your bases. Outside of IT, quality and effectiveness are judged by delivered results. For instance, it's widely accepted that a Lexus is a higher-quality car than a Corolla, which is also manufactured by Toyota. It's the cars themselves that warrant the distinctions. The Lexus' superior engineering, materials, and workmanship enable producing the higher quality; but it's the result, not the practices creating the result, which is recognized as higher quality.

Some other automaker conceivably could be using comparable engineering, materials, and workmanship but still might not be producing a high quality car. Some of the differences might be in the realm of beliefs, customs, knowledge, skills, and work habits, especially as promulgated through management practices. Companies that repeatedly produce excellence know their REAL Process includes all these factors.

On the other hand, the quality of one's software is not a criterion for being assessed highly by some of the high-overhead branded approaches. Rather, assessments presume quality will result from following their model process procedures and activities; but the assessments do not evaluate how well those procedures and activities are performed. Moreover, the branded models and assessments generally do not address beliefs and customs, including management practices, which indeed are part of the REAL Process affecting the results.

Furthermore, a seldom-recognized dirty not-so-little secret of most branded big ticket imposed processes is a *giant loophole—management practices generally aren't addressed*. So, after all the worker bees have jumped through all the formal procedure busywork hoops, which may in fact have been helpful producing better software, management continues to make its decisions in the same old, same old ways.

The REAL Process of these branded approaches is the flawed presumption that position automatically produces wise decisions commensurate with the branded process' disciplines. Perhaps you too have heard expressions such as, "Ship it, we'll fix it later." and "Deadline. Deadline. Deadline." Using branded high-overhead process improvement approaches doesn't ensure such expressions don't continue to reflect management's REAL Process for making decisions. Generic REAL process improvement includes identifying and addressing beliefs, customs, and management practices that affect results.

2 Skills Enhancements

In many instances, improvements can be achieved quickly and relatively inexpensively by helping people enhance relevant skills. While most such training tends to focus on domain knowledge, methodologies, techniques, or tools, one should not overlook the potential benefits of enhancing more generic skills such as simply improving typing speed and accuracy.

Too often, though, the REAL Process is that training provides little real value, not because the training is no good; but because training becomes an end in itself rather than a means for improvement, doled out with no real plans to utilize its lessons; or, as many seminar attendees so frequently say, "They won't let us use it." Lessons learned in training fade quickly when they are not used, which is inevitable when attendees return to the same old environment which doesn't know the lessons.

The biggest cause of training's non-use tends to be the trainee's manager who believes that s/he doesn't need the same training. Sometimes there is fear of appearing less than all-knowing. Often objectively superior techniques are resisted because they differ from those the manager has relied upon and been rewarded for.

Probably the largest skills gap impacts are in the areas of leadership and people-oriented soft skills, primarily among management who don't recognize their own deficiencies or

by Robin F, Goldsmith, JD
robin@gopromanagement.com www.gopromanagement.com
This article will appear in ASQ Software Division's *Software Quality Professional* magazine

are in denial about them. Time and again capable workers' performance is reduced by poor management and supervision, which not coincidentally is a leading cause of good workers' voluntarily leaving organizations.

3 Management Practices

Different from enhancing the skills of managers is implementing specific "good" management practices that seemingly can be applied across the board to economically achieve improvements in all projects. Such initiatives generally are undertaken without the analysis and identification of needs which is part of generic REAL process improvement. However, it's common for such initiatives to address very REAL, albeit unacknowledged, weaknesses which afflict practically every organization.

The old adage, "You can't manage what you can't measure," is attributed to many sources but remains relevant regardless. Consequently, introducing measurement/metrics programs is often the chosen form of management practices improvement. In many organizations, the REAL Process is a woeful lack of valid, reliable, and meaningful measurements. Of course, one of the main reasons for such deficiency is failure to identify the REAL Processes producing results, since measures must be of the REAL Process to meaningfully guide improvement.

Unfortunately, the REAL Process in most organizations is that most measurement programs fail. The biggest obstacle is the often-justified fear of workers that measurements they make will be misused punitively against them. Resistance also is common when those asked to make the measurements find that doing so takes considerable time away from their regular responsibilities without apparent benefit to them personally. In addition, measurement activities quickly cease when it is perceived that the measures don't lead to relevant actions, which often is due to measuring everything without much idea what to do with the measures or overgeneralizing from inadequate data, frequently to justify prior decisions.

Key REAL Process measures that are especially likely to aid improvements include identifying the number, magnitude, and financial impact (such as costs to find and fix) of defects relative to engineering practices employed and life cycle phases of origin and detection. One of the biggest measurement challenges involves quantifying work output. Techniques such as counting function points, which provide programming-language neutral measures of program/system size, can be especially useful. However, size alone does not equate to value, which ultimately must be stated in financial terms and also is very challenging for most organizations. The REAL Process is that most Return on Investment (ROI) calculations are specious because they fail to quantify intangibles[3].

Other management practices that are potentially good candidates for improvement relate to personnel actions, such as hiring, firing, delegation, performance evaluation, and related reward mechanisms.

by Robin F, Goldsmith, JD
robin@gopromanagement.com www.gopromanagement.com
This article will appear in ASQ Software Division's *Software Quality Professional* magazine

ENGINEERING "GOOD" PRACTICES

The big branded models don't get into the "how" engineering specifics. That's not their thrust, which is fine, except that they presume dictating that a process should address issues such as requirements and quality thereby assures it will be done well. In fact, many organizations are woefully deficient in these areas. Darn those presumptions.

An alternative low-overhead improvement approach that many organizations use is simply to bypass "software process improvement" initiatives and proceed directly to implementing presumed engineering "good" practices. By skipping the identification and analysis of the REAL Process, implementing "good" engineering practices incurs even less delay and overhead than with generic software process improvement. Of course, as with any imposition, there's always the possibility that the imposed "good" practice is not really an improvement and could even be a step backward, which is especially likely if the organization implements and executes the practices poorly.

On the other hand, and emphasizing this big "if" qualification regarding proficiency, implementing reasonable "good" practices, such as the four described below, probably will pay off for most organizations, because the areas these practices address tend to be both important for all software organizations and deficient in many.

4 REAL Requirements

The highest payoff undoubtedly can be obtained by learning to discover requirements, since everything else ultimately depends upon requirements adequacy. It's virtually universally recognized that the cost of fixing a requirements error increases dramatically the later the error is corrected. It's somewhat less widely recognized that around two-thirds of errors originate in requirements. Since requirements errors often are fairly old before they are detected, the cost of fixing requirements errors represents an even higher disproportionate share of total development cost. Thus, reducing the number and impact of requirements errors is an obvious top candidate for targeted process improvement.

Unfortunately much of the IT industry's conventional wisdom and widely-accepted requirements models are flawed[4] in ways that are likely to cause well-intentioned requirements improvement efforts to achieve far lesser benefits than they could and should. Additional obstacles to meaningful improvement are lack of awareness of and resistance to recognizing these flaws, often accompanied by the additional flawed conventional wisdom that requirements change so rapidly and frequently that's it's not worth making the effort to define the requirements.

Several prominent methodologies start with such presumptions, rationalizing their inability to discover requirements by declaring it impossible to discover requirements, which takes the onus off them, and then essentially turns not discovering the requirements into a virtue, which becomes the current version of the old cartoon, "You start coding, and I'll go find out what the requirements are."

by Robin F, Goldsmith, JD
robin@gopromanagement.com www.gopromanagement.com
This article will appear in ASQ Software Division's *Software Quality Professional* magazine

Much of the weakness of conventional wisdom and models originates because they ordinarily use the term "requirements" to mean the human-defined requirements of the product, system, or software that is intended to be created. The conventional wisdom model is that "business requirements" are high-level and vague, often just goals and desired benefits, and decompose into detailed product/system/software requirements. Emphasis is primarily on the writing of the requirements so they are clear.

The conventional model is flawed in thinking that the difference between business and product/system/software requirements is simply a level of detail, where a requirement that's vague and high-level is a business requirement, whereas a requirement that's detailed is a product/system/software requirement.

The difference is not the level of detail. REAL business requirements are qualitatively different from product/system/software requirements. REAL business requirements are from the perspective of and in the language of the business/user/customer/stakeholder. They are conceptual and exist within the business environment, which means they must be discovered. REAL business requirements are business deliverable *whats* that provide value when delivered. There are usually many possible ways to accomplish (or satisfy, meet, or deliver) the REAL business requirements.

In contrast, product/system/software requirements are really high-level design of a human-defined product or system which presumably is one of those possible ways presumably *how* to presumably accomplish the presumed REAL business requirements. These are often also called functional requirements or functional specifications, since they frequently are phrased in terms of the external functioning of the product/system. Humans define a "specification," which is a synonym for design. A design does not have to be technical engineering detail. A product or system solution is a design too, albeit high level. The product/system/software requirements provide value if and only if they actually accomplish REAL business requirements.

REAL business requirements *whats* do not decompose into product/system/software requirements *hows*. *Hows* are responses to the *whats*. All the *how* detail in the world won't make up for not know the *whats* that the *hows* must meet.

Conventional wisdom is that creep, changes in requirements after they presumably have been defined, is mainly due to product/system/software requirements that are not sufficiently clear and testable. While product/system/software requirements clarity and testability are important, much of creep occurs because the product/system/software requirements, regardless of how clear and testable they are, do not satisfy the REAL business requirements, which occurs mainly because the REAL business requirements have not been defined adequately, mainly because conventional wisdom and models focus all their attention on what they erroneously think are *the* requirements—the product/system/software requirements of the product or system they intend to create.

by Robin F, Goldsmith, JD
robin@gopromanagement.com www.gopromanagement.com
This article will appear in ASQ Software Division's *Software Quality Professional* magazine

To make meaningful significant improvement in requirements definition, one's development process must first discover the REAL business requirements. This is not easy, partly because of having to overcome long-standing flawed conventional wisdom presumptions and partly because it is difficult to discover the REAL business requirements. The improvement starts with a changed mindset, realizing that REAL business requirements don't change nearly so much as the *awareness* of them and that REAL business requirements must be from the perspective of, and in the language of, the business/user/customer/stakeholder.

To get the requirements right, it's also essential to realize that one cannot merely expect the business/user/customer/stakeholder to dictate them. One doesn't just gather requirements, because most REAL requirements are not readily apparent. Rather, skilled analysis is needed to discover the REAL business requirements.

My powerful Problem Pyramid™ tool is especially helpful for identifying the REAL problem, challenge, or opportunity which provides REAL value when solved/addressed and the REAL business requirements which when delivered solve/address the problem/challenge/opportunity. My requirements definition seminar[5] and book, *Discovering REAL Business Requirements for Software Project Success*[6], provide further direction and practice using the Problem Pyramid™ and other relevant REAL business requirements discovery techniques.

5 Reviews

Code reviews are widely regarded as the single most economically effective way to detect defects. It's estimated that code reviews can find defects in as little as one-tenth of the time it would take to find the same defects with dynamic code execution testing; and code reviews can find many errors, especially those affecting security, which dynamic test execution essentially may never find. Yet, few organizations review code.

Resistance to code reviews is likely due in part to lack of awareness of the technique's power and in part to cultural resistance of developers, who as a group have somewhat of a reputation for discounting the need to test their code in any way, but especially by reviews which subject their code to public scrutiny. With suitable leadership, including relevant metrics, code reviews offer a very low-overhead way to detect and remove more software defects in less time and at lower cost.

In contrast to code reviews, many organizations do review requirements and/or designs. Catching errors early with reviews helps prevent problems which would be much more difficult and costly to fix if they were not found until later. Despite the obvious value of such reviews, most organizations undoubtedly overestimate their effectiveness and don't realize how weak their reviews really are. For instance, most organizations use only one or two techniques to review requirements and designs, whereas we have more than 21 ways to review requirements and more than 15 ways to review designs, many of which are far more powerful than commonly-known review techniques[7].

by Robin F, Goldsmith, JD
robin@gopromanagement.com www.gopromanagement.com
This article will appear in ASQ Software Division's *Software Quality Professional* magazine

6 Reuse

Reusing software artifacts can save time otherwise needed to recreate such artifacts. Programmers regularly apply a number of reuse techniques, such as accessing web services, incorporating components and subroutines, including copied standard pieces of code and file/database table descriptions, and copying and pasting one's own code from program to program.

It's far less commonly recognized that other types of software artifacts also can be reused. For instance, many organizations utilize templates of various types without necessarily realizing they are a form of reuse. Certain requirements, designs, and user instructions may pertain to more than one system and be reused rather than recreated. Test cases often can be reused, as can a little-known high-leverage technique called test design specifications[8].

While effective software professionals frequently reuse artifacts they have created themselves, there are two common obstacles to reusing artifacts created by others. First, people have to know what is available for reuse, how to find the things that are available, how to reuse them, and how to create additional reusable artifacts. Having an understandable way to organize the artifacts is essential and relatively easy to accomplish.

The second, more difficult obstacle to overcome is the prevalent "not invented here" attitude, whereby people prefer to use only things they have created themselves. Leadership and education can help address such attitudes, to an extent.

7 Proactive Testing™

Pre- and post-implementation defects are a major source of user dissatisfaction, system ineffectiveness, and corrective costs. Software testing is the primary means of detecting defects and takes up about half of typical software projects. Consequently, testing is a prime candidate for low-overhead improvement.

Traditional testing improvement approaches generally focus on encouraging earlier tester involvement, applying test design techniques which more systematically and therefore more thoroughly identify executable test cases, and analyzing risks so that the test cases dealing with the higher risks are executed more. Such testing improvements do pay off, but few people recognize that benefits often are lower than they could be because traditional testing practices tend to be reactive. That is, traditional testing often comes too late and is largely reacting to the software that's been developed and already contains all the errors.

In contrast, my Proactive Testing™ methodology[9] applies a variety of special techniques and approaches that anticipate more of the often-overlooked potential problems so they can be addressed earlier, when they are easier and cheaper to fix.

Rather than being resisted as an obstacle to delivery, Proactive Testing™ wins advocates among managers, developers, and users. A key Proactive Testing™ concept emphasizes finding WIIFMs (What's In It For Me), testing techniques that others find actually help them deliver better systems quicker, cheaper, and with less aggravation.

A related Proactive Testing™ concept is *letting testing drive development*, which doesn't mean that testers become the bosses of developers. Instead, it means identifying information from a testing perspective that developers find helps them do their job. For instance, rather than focusing primarily on low-level test cases, Proactive Testing™ starts with master test planning risk analysis which identifies larger risks that ordinarily are missed and turn into showstopper errors. Beyond merely testing more for these high risks when they are hard to fix, anticipation enables building software in different ways which can catch the high risks far earlier when they are much easier to fix and prevent extra rework due to building software which otherwise would have to be rebuilt.

Proactive Testing™ also includes and goes well beyond test-driven development techniques popularized by Agile Development, truly putting those methods "on steroids."[10] Test-driven techniques include coding automated unit tests before coding the program. The program is deemed to be correct when it passes the unit tests, which are supplemented and re-executed whenever the program is modified. Also, user-defined "acceptance tests" are automated to exercise more functionality (comparable to integration tests) than the program unit.

Along with pair programming, which provides ongoing review of code as it is developed, these test-driven techniques can produce higher-quality code by assuring it is subjected to considerably more and earlier quality assurance and testing than ordinarily occurs with traditional development. While certainly an improvement, tests defined by the developers or users are unlikely to be nearly so thorough as presumed.

Proactive Testing™ test planning and design techniques can identify numerous often-overlooked major and minor test conditions, which can reduce risks by alerting the developer to be sure the conditions are addressed, without necessarily incurring the more time-consuming optional effort of creating executable tests for these additional test conditions. Furthermore, rather than simply reacting to the system being built, Proactive User Acceptance Testing™ takes an independent approach[11] to guide defining acceptance tests from a truly business/user view which confirms that what the development process thought it should be developing in fact is what the business really needs. Defining Proactive User Acceptance Criteria not only provides a basis for prioritizing and structuring the executable user acceptance tests, but it's also one of the more powerful of our 21 ways to review requirements, spotting wrong and overlooked requirements.

by Robin F, Goldsmith, JD
robin@gopromanagement.com www.gopromanagement.com
This article will appear in ASQ Software Division's *Software Quality Professional* magazine

Summary

Low-overhead generic software process improvement offers attractive benefits compared to high-overhead branded formal process improvement initiatives. The key is identifying and then directly addressing only high-payback issues within the REAL software development process. Short-cut approaches offer likely benefits by bypassing analysis and proficiently implementing presumed "good" management and engineering practices, such as training, metrics, discovering REAL, business requirements, reviews, reuse, and Proactive Testing™.

References

1 ® CMM and Capability Maturity Model are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.
2 Paulk, Mark "'Practical' Software Process Improvement," Keynote Presentation at First International Conference on Software Process Improvement (ICSPI), November 19, 2002, College Park, MD.
3 Goldsmith, Robin F., "ROI is Deceptive Without REAL Requirements and Quantified Intangibles," Requirements Networking Group featured article, March 20, 2007, http://www.requirementsnetwork.com/node/735.
4 Goldsmith, Robin F., "Conventional Requirements Model Flaw Misses REAL Business Requirements," Requirements Networking Group featured article, February 13, 2007, http://www.requirementsnetwork.com/node/683
5 Go Pro Management, Inc., *Defining and Managing User Requirements* two-day seminar, www.gopromanagement.com.
6 Goldsmith, Robin F., *Discovering REAL Business Requirements for Software Project Success*, Artech House, 2004.
7 Go Pro Management, Inc., *Reviewing Requirements and Designs(Testing Early in the Life Cycle)* two-day seminar, www.gopromanagement.com.
8 Go Pro Management, Inc., *Developing Reusable Test Designs* full-day seminar, www.gopromanagement.com.
9 Go Pro Management, Inc., *Proactive Testing™: Risk-Based Test Planning, Design, and Management* two-day seminar, www.gopromanagement.com.
10 Goldsmith, Robin F., "Proactive Testing™ Puts Agile Test-Driven (and Other) Development on Steroids," presentation at Software Test & Performance Conference, October 4, 2007, Cambridge, MA.
11 Go Pro Management, Inc., *Proactive User Acceptance Testing*™ full-day seminar, www.gopromanagement.com.

About the Author

Robin F. Goldsmith, JD has been President of Go Pro Management, Inc. consultancy since 1982. He works directly with and trains professionals in business engineering, requirements analysis, software acquisition, project management, quality and testing. He partners with ProveIT.net in providing ROI Value Modeling™ tools, training, and advisory services. He is author and Course Director for ASPE's two-day seminar, *Managing Real-World Processes and Projects with Metrics*. He also presents a variety of seminars and consulting based on his REAL business requirements and Proactive Testing™ risk-based methodologies for delivering better software quicker *and* cheaper.

Previously he was a developer, systems programmer/DBA/QA, and project leader with the City of Cleveland, leading financial institutions, and a "Big 4" consulting firm.

Author of numerous articles and the recent Artech House book *Discovering REAL Business Requirements for Software Project Success*, and a frequent speaker at leading professional conferences, he was formerly International Vice President of the Association for Systems Management and Executive Editor of the *Journal of Systems Management*. He was Founding Chairman of the New England Center for Organizational Effectiveness. He belongs to the Boston SPIN and served on the SEPG'95 Planning and Program Committees.

Mr. Goldsmith Chaired BOSCON 2000 and 2001, ASQ Boston Section's Annual Quality Conferences, and is a member of the ASQ Software Division Methods Committee and the IEEE Software Test Documentation Std. 829 revision Committee.

He holds the following degrees: Kenyon College, A.B. with Honors in Psychology; Pennsylvania State University, M.S. in Psychology; Suffolk University, J.D.; Boston University, LL.M. in Tax Law. Mr. Goldsmith is a member of the Massachusetts Bar and licensed to practice law in Massachusetts.