Craig S. Mullins

Return to Home Page

May 1995

System Development Magazine

Producing Quality Software

by Bernard S. Klopfer and Craig S. Mullins

Total quality management. Quality initiatives. Zero defect software. Six sigma programs. These terms are infiltrating the once isolated bastion of software development. Obviously, the overwhelming drive toward software quality is upon us and there is no turning back. But what overriding factors must be understood before embarking upon a quality program within the realm of information technology? Why is a quality program important and, even more to the point, haven't we been striving to produce quality software already?

This article will examine aspects of software quality from both a humanistic perspective and a technological perspective.

Change Drives Quality Goals

The changing corporate business landscape demands that software developers pursue excellence in the eyes of broadly defined customers. The concept of quality applied to software products and end-user applications has traditionally been defined as reduction or absence of defects. This is no longer a sufficient definition. Yourdon seems to suggests in "The Decline and Fall of the American Programmer" that the quality of software is defined by the development process but most importantly by the people that manage and are managed within the process.

The human element is the most frail component within the spectrum of software quality. This is so because each individual has his own set of concerns that impact his ability to manage change. To manage software quality one must first be able to manage the people that produce that software.

Examination of the people factor in relation to quality reveals three typical behaviors among software developers and managers that serve as impediments to the production of quality software.

"Wyle E. Coyote - Super Genius"

We can all remember that poor, self-proclaimed genius from the Warner Brothers cartoons of our youth—Wyle E. Coyote. It seems he had a marvelous contraption for every occasion. The "Wyle E. Coyote - Super Genius" behavior as applied to the data processing professional is best reflected by a development programmer who receives the design specifications for an application and says: "I can develop a better system with more features, slicker algorithms, and fewer modules—each with no more than 256 lines of code." This type of excessive creativity often results in late and over-budget delivery of the system. Remember, Wyle E. Coyote, for all of his cleverness, never met with success.

When a system does not meet the deadline, resultant perception of the system is lessened because one of the greatest quality concerns among end users is on-time delivery. (Typical end user sentiment: *What difference does it make if it will work wonders when I get it, if I can not use it when I need it?*) Software development organizations should not define software quality on behalf of their users. Reigning in the free-wheeling attitude of the *super genius* programmer may be difficult, but it is mandatory if software quality is to be a major imperative at your shop.

"Produce or Perish"

Another typical behavior is a direct result of the precarious times in which we live. With corporate downsizing, early retirement programs, layoffs, and the like, employees are apt to believe that if they do not give more than 100% they will be terminated. Analogous to success in academia, programmers (and managers) that exhibit this "Produce or Perish" behavior approach application development as a means of survival. The typical reaction to application requests reflect a focus on the delivery of the end product, not the process. The classic cartoon depicting a room of developers and the manager saying "Everybody start coding and I'll go find out what they want" while humorous, is reflective of this type of behavior. When challenged with a short delivery time frame the immediate reaction is to ignore the methodology, if one exists, and start producing the product without regard to improving quality through improving the process.

But, methodologies exist for a reason. They are there to provide a controlled approach to application development—one in which steps are delineated that will result in the optimal production of the required application system. If the methodology can not be used for your typical application development needs, then:

- either the methodology is flawed and should be replaced with a more appropriate methodology for your situation, or;

- the decision to subvert the methodology is flawed and should be re-examined

The bottom line: do not let an over-zealous lurch to quicken the delivery of an application result in having the process that was initiated to produce the application properly (the methodology) be ignored. Methodology is important, but as we will discuss later, it must not be dogmatic and inflexible.

"A Room Without a View"

Sometimes, software developers feel that they operate best in isolation. This occurs when application development becomes a finite project performed in a vacuum rather than as part of an ongoing process. Software developers have the perception of being so busy that they are unable to see their work as part of a continuous process. Consequently, they do not take the time to apply techniques or lessons they have learned. When the opportunities for improvement have not been institutionalized or at least formalized and recorded, the successes that occur are more often than not based on chance.

It is important that everyone on the software development team act, and be treated like part of, a team. Failure to do so will greatly impede the resultant quality of the system. A successful approach to initiating an attitude of teamwork can be achieved by following the 3C approach— communication, cooperation, and clarification.

- ***Communicate*** effectively at all levels within the group, from the top down (managers to subordinates) and from the bottom up (subordinates to managers).

- Stress that it is necessary for each member of the team to ***cooperate*** with one another. No member of any team can afford to say "that is not in my job description." Of course, cooperation should not be viewed as a synonym for off loading work to co-workers.

- And finally, ***clarify*** all matters which are confusing. Do not blindly attempt to develop what you do not understand. Once again, clarification should be implemented in both directions—top down and bottom up.

By embracing the 3C approach, the "island unto myself" attitude can be eliminated and the possibility for quality software will be heightened.

Lack of Procedural Application

Implied in the behavioral characteristics of a software organization is that the a formalized development process is severely lacking. Ironically, many good software engineering principles and methodologies are well-known and well-established. It is the application of these principles that is lacking because the techniques and tools are seldom clearly defined and institutionalized.

A study of software engineering practices at major companies conducted by the Software Engineering Institute (SEI) of Carnegie Mellon University in Pittsburgh, PA, shows the state of the development process. Refer to Figure 1. According to this study, a mere 1% of organizations have instituted a formalized software development process.

Figure 1. The State of the Development Process

The ideal goal for software development organizations should be to reach the point where the development process is closely managed and measured quantitatively, with improvement fed back into the process. SEI reports that none of the companies surveyed had achieved this level of development in their software engineering.

As startling as these statistics seem, the most surprising aspect of software quality has been the level of tolerance by the consumer.

Traditionally, it was widely accepted that an IS department or software vendor would typically over promise and under deliver. The impact of this type of delivery of products to the consumer was generally negligible because the IS departments and software vendors were the only game in town.

Satisfy the Customer

Software users today are more sophisticated and demand more in terms of functionality and ease of use than they did a mere ten years ago. The expansion of their knowledge has been drawn from a broad base of experience with commercial PC software products. The current trend toward demanding quality is a direct by-product of the personal computer revolution. End users have become less tolerant of spending a lot of time or money to produce applications or receive software products, both in absolute terms and regarding predictability. One need only look at the broad acceptance of client/server and departmental computing to realize that end users have a great desire to participate in and control the software development process. It is not a far stretch to state that the broken promises and high costs of days gone by have caused this scenario.

The key to establishing quality is defined by ***customer satisfaction***—whether that customer is inside or outside the firm, a small account or a major client, a small department or large department. When faced with intense competition, ***customer satisfaction*** is the only definition of quality that matters.

Consider the following: Given a choice between a Dodge Stealth that breaks down 50% of the time and a Toyota Corolla that is 99% reliable, which car would you choose?[1] What if the further stipulation were applied that this one car would be your only car? Some would choose the Stealth because of its style and sportiness. Others will select the Corolla because of its dependability. The point is that there is no globally correct answer. To arrive at the correct answer one must understand the customer and cater to his wishes.

Instituting a Flexible Methodology

Software development productivity has improved over the past 20 years, but perceived software quality, customer satisfaction, has not kept pace. The software development process must be guided by a comprehensive software quality improvement program.

Mandating what may be termed a "commercial methodology" which incorporates idealized procedures and methods has proven to be ineffective and viewed as a nuisance by all participants. The need for procedures and structure is paramount to the success of any type of software development but is not generally reflected as part of a commercial product. What is of primary importance for the adoption of a methodology is an agreement on a basic philosophy that underlies and supports the motivational factors that drive the product group. Additionally, it is important that strict adherence to the "written word" be absent from the methodology, a methodology must be flexible.

The rigorous methodologies that are in place within most organizations are not successful directly because of the rigor. Shifting the understanding of those who must adhere to the methodology is crucial. This shift can not be achieved without accomplishing two objectives:

- Modify your methodology to make it less stringent. Most commercial methodologies are procedural in nature and require strict adherence to the letter of its processes. By reducing the "*A is followed by B which is always before C*" nature of the methodology, it will be more generally applicable to your software development tasks.

- Educate the developers on the methodology and the purpose for the methodology. Without stating a purpose your methodology will usually fail regardless of how flexible it is. Because the developers will view it as bureaucracy and a hindrance. With a noble goal, however, a flexible methodology has a chance to succeed.

Streamline the Perception

One key area of focus for a management group whose mission is to improve quality, is to first establish an internal definition of quality that adheres to the goal of customer satisfaction. While this may at first appear to be a simple task, the perception of what represents quality from the perspective of a development person probably differs greatly from the perspective of a support individual. This in turn will vary from the perspective of the end user, the manager, the analyst, etc.

Coming from a developers background, one's perception of quality may be primarily concerned with improved execution and efficient memory utilization, slick algorithmic solutions, and code efficiencies. In the eyes of the developer, if a program or product achieves these types of goals, a quality product is delivered. In fact, contests exist where developers submit C code that is purposely cryptic but must still perform a function.

This is where the divergence of support or end-user and development's perception of quality software occurs. It is not generally thought to be development's responsibility to provide direct consumer support of a software product, thus the perception of module/software quality is controlled by the individual developers ideology. Whereas, the perception of quality from technical support staff or end-user is quantified by the gross number of problem calls received or made as a function of their area of responsibility.

Quality as a corporate mission propagates more involvement in the realm of Quality Assurance ideology and the perception of what quality software is, begins to change.

Quality reflects reliability. We expect it as consumers of goods and services and thus we should expect and understand the concept in relation to software. Quality software, therefore, should reflect reliability. Reliability propagates improved perception and ultimately customer satisfaction.

The Teamwork Resolution

In order to respond to the demands of the software consumers, be it an end-user or customer, a definite progressive procedural structure associated with development and support that reflects a commitment to quality is necessary. An enhanced philosophical vision that the product itself, and all problems found as a result of support service functions, belong and become the responsibility of the group as a whole. This will initiate an environment in which everyone works toward the common goal of reliability and quick problem resolution. Too often it is evident that blame is applied to the different factions within a software organization when the resultant product is of poor quality.

Application of a generalized quality philosophy is easier to write about than to implement and most "Quality Assurance Methodologies/Programs" do little to establish an underlying philosophy. What we find when we institute a "Quality Program" in an organization, or are evaluated by "quality experts", is that the various tools and procedures recommended and associated with these "Programs" reflect idealized concepts that are at the very least difficult to implement. Additionally, the efforts associated with the application of these "Quality Programs" do not take into account the realities of the re-education, philosophy change, and expected management buy-in.

The resolution to this often haphazard attempt to force quality through adoption of idealized concepts requires careful review and evaluation by the people that can direct and implement the type of underlying philosophy mentioned above.

Adoption of a successful quality program begins with two basic ideas:

- work to define what quality means, and;
- establish a baseline philosophy that centers around a pride in work/ownership in relation to a product.

Once these goals have been accomplished a means of measurement can be established to evaluate where the product meets or falls short of the quality definition. Indeed customer enhancement requests, or support call figures can be used to measure customer satisfaction. Additionally, the areas of "lesser quality" will become focal points of the individuals responsible for those areas and the motivational factors associated with ego will play a key role in improving quality. Adoption of this philosophy and defining quality are key focus areas for the initiation and continuation of improved quality.

Corporate Acceptance of the Quality Initiative

Sometimes the process of instituting quality measures is deemed to be a waste of time. Of what value is a 50% decrease in defects if it comes at the cost of a 50% decrease in production? Has quality actually increased in this scenario?

These are reasonable concerns. However, they are not necessarily valid. The first approach to take when confronted with this type of logic is to ask the skeptic to define “productivity.” Too often, managers base their definition of productivity on new development only. If a quality program reduces productivity in terms of new development, it will most assuredly increase software maintenance productivity. Up-front quality measures always result in reduced post-implementation defects. But the productivity of maintenance is rarely, if ever, measured. How often have you heard: “The project was delivered on time and within budget.” But, how often does this statement get made when the development staff knows that there is an extensive list of post-implementation “enhancements” yet to be made? A mindset needs to be established whereby productivity measurements are undertaken for all software engineering activities. After all, what is new development but radical maintenance?

However, even after dissecting the definition of productivity, it is still true that any type of change is accompanied by a learning process which will initially result in reduced output. The introduction of new quality concepts is no exception to this rule. If communication and a rigorous training program accompanies the quality initiative, then the initial decrease in production can be mitigated. The communication aspect is as important as the training aspect in lessening the impact of production falloff. To achieve success, the following messages must be effectively communicated from management to staff:

- The quality initiative is not a reaction to the current quality of the systems being developed; instead it is part of the continuing learning curve associated with the relatively new discipline of computer science. Computer science has existed for less than 40 years. This is in stark contrast to other sciences, many of which are thousands of years old. Moving from an application development environment in which quality assurance is not a part of the development methodology to one in which it is can be likened to the discovery by astronomers that the Earth orbits the Sun, and not vice versa.

- The quality initiative is being instituted so the company can function better, thereby enabling everyone to benefit when the company more effectively competes in the marketplace. It is not a ploy to reduce staff, enforce overtime, or simply "shake the trees." In fact, it may have the opposite effect. When things are done properly the first time, less overtime will be required, defects will decrease, thereby causing the desire to shake things up to decrease.

- Following a quality program will cause developers to feel better about their job. Just knowing that the project is being rigorously tested and measured should instill a "pride" of authorship in the developers that will further enhance overall software quality.

When everyone understands that the quality initiative is not punishment for past "sins," but a new and better way of doing things, initial acceptance increases thereby minimizing the "growing pains" associated with the new methodology.

Of course, every developer who is targeted to participate in the quality initiative must be trained in the methods of being instituted to assure quality software. Ideally, this should include all developers. This is where management buy-in to the quality objective is essential. Without management commitment, training will be lacking. And who can fault developers for failing to deliver quality products based upon the approved methodology if they have not been trained?

As the quality program becomes accepted and its components become understood, production will once again increase to past levels or better, and this time, not at the cost of quality.

Promoting Quality Without Management Buy-In

The discussion of procedural improvements and the adoption of a structured methodology implies a supporting environment within a software organization that allows, or at least condones the associated activities. In practice the vast majority of software organizations reflect a significant amount of “lip-service” towards the cause, but little or no action. An underlying truism is associated with this type of approach to software development; “there is never enough time to do it right, but always enough time to do it over”. The major commercial software development companies spend millions of dollars on product support, fix releases, and patches for their customers. Most corporate data processing shops have 24-hour application support for abends occurring after normal business hours. While the actual percentage of total dollars spent on these activities can not all be directly tied to problems that arise from faulty software, faulty software can “cost” significantly. This cost is associated with customer and end-user perception of commercial and corporate applications.

So what can be done? How can quality software be promoted without management cooperation? As with most solutions to problems, the answer lies with the individual. The individual, in this case, is the manager that can influence or direct their teams with relative autonomy. When an individual makes a decision to travel down a particular path for self-improvement, goals and milestones are set to accomplish that achievement. The same ideology applies to improvement in process maturity associated with software development. Listed below are some suggestions for instituting process improvements that require minimal effort and reap significant benefits:

1. Implement process improvements as small step, procedural changes that provide more structure to existing processes.
2. Empower members of the project team with responsibilities associated with the implementation of the application of “structure.” Get the team involved in the process.
3. Establish and use inexpensive project management tools to project, and track tasks and resources associated with all or one particular phase of a project’s life cycle. This will establish baseline statistics and improve estimates for future projects.
4. Institute task tracking by linking time reporting to task completion through the use of individualized tasking sheets or spreadsheet links to project management tools.
5. Adopt metrics associated with establishing these techniques:

- Establish baseline measurement of an easily-definable result of process (i.e.; end user satisfaction qualified and measured by raw number of problem reports)
- Institute structured techniques, then measure and analyze results of the process
- Share the measurements and results with the team
- Evaluate applied structure and improve

6. Establish a consensus understanding of quality products and results.
7. Encourage ideas for improvements from the team. There is much to be said for “pride of authorship.”

These steps result in the application of procedures—a methodology to facilitate the quality improvement process. In short, the basis for the success of any type of procedural/process improvement begins with the participants.

Enabling Quality

The most important tools that can be used to promote the continuous improvement of the process are quality metrics based on an organization's major quality issues. These measurements provide a benchmark; feedback from end users and direction for improvement for software development teams.

Specifically, some attributes of quality that usually contribute most to customer satisfaction are predictability, business impact, appropriateness, reliability and adaptability.

- Predictability means knowing and controlling the risk factors involved, such as delivery date, cost and resources needed.

- The business impact dimension of quality relates to whether or not an application provides a significant benefit to the organization, even if it is not technically elegant or efficient.

- An appropriate approach will provide a solution that is suitable to the business and technical problems which end users want addressed.

- Reliability concerns whether or not the system will work for its users when they need it.

- Adaptability means that the system and support are able to change cost-effectively to meet the evolving needs of the end users and the users' customers.

By breaking each of these quality issues down into their key components and then formulating a measurement system based on those components, managers can begin to get a handle on the software development process. The key, then, to a software quality management program lies in whether or not the application development process can be measured. If a process cannot be measured, it cannot be understood, controlled or improved.

There is a hidden danger when developing quality metrics for your organization, though. If the metrics do not truly reflect what is indicative of quality for your organization, then quality will be ill-defined and difficult to achieve. For example, consider our discussion of customer satisfaction earlier in this article. If the satisfaction component is absent from the metrics, then a key part of quality will be ignored. Remember, quality must be defined by those who use the end product, in this case the software. Yet, metrics that define true customer satisfaction are difficult to obtain. Usually, one must be content with metrics (such as for the concepts listed above) that define portions of the customer satisfaction equation.

Using Technology to Augment Quality

Although much of this article may seem to be railing against technology, this is not the intent. Technology for technology's sake will never result in quality systems. However, the appropriate adoption and adaptation of technology can provide increases in overall software quality.

Consider, for example, the current trend toward object orientation (OO). The biggest selling point of OO is the increased reusability it offers. Reusability occurs because OO concentrates on objects instead of mere data elements. An object is defined in terms of both its state and it behavior. The data and the operations that can be performed on that data are encapsulated into a central store called an object.

How does this increase reusability? The underlying components of an application are hidden inside the objects comprising that system. It is not possible for an external agent to modify an object's state. The external agent must send a message to that object, invoking a method internal to that object, thereby altering the object's data. The object is essentially a plug-and-play piece of software. In fact, OO development environments are characterized by their class libraries which contain the reusable objects.

It becomes obvious that incorporating an OO approach into the software development methodology will be of great benefit. No longer will it be necessary to reinvent the wheel for each new application system. With OO techniques, objects in class libraries become building blocks. If enough objects are pre-defined, the construction of an application becomes a simple matter of inter-connecting the pre-existing objects to function as desired.

Additionally, maintenance of OO systems is also easier. Because code is embedded within an object, simply changing the code once in the only place it exists causes the desired change to be incorporated into every system that uses that object.

However, one must approach technological advances with caution. Simply adopting OO approaches for new software development does nothing about current legacy systems which were written using many different approaches. Likewise, technology must not be approached as a silver bullet. The history of software development is characterized by a long list of panaceas that proved to be anything but the ultimate solution. Remember structured programming, 4th generation languages, CASE, and JAD? All of these techniques were going to revolutionize software development and put scores of maintenance programmers out of work. It didn't happen then, and it isn't likely to happen any time soon.

Synopsis

Instituting quality procedures for software development is impossible without a thorough understanding of what that process entails. This understanding is not rooted solely in the technological aspects of software development. It must encompass the “people” aspect as well. Keep the following key concepts in mind as you institute a quality initiative at your shop:

- Understand the needs of your customer and meet them without being biased by the tangential desires of your staff.
- Understand the needs of the software developers and institute policies to involve them in the quality initiative. Remember the three personality types and be prepared to deal with them.
- Remember the 3C approach: communicate, cooperate, and clarify. Without all three, a project will be doomed to failure.
- Inform, instruct, and empower the individual to operate within the quality program.

- Be firm, yet flexible. A methodology should not be so restrictive as to tie the hands of the developer. Yet, it must not be so “loose” as to be ineffective.
- Establish metrics to gauge the effectiveness of the quality program on the software that is developed.
- Incorporate technological advances into your quality framework without viewing them as the answer to all of your needs
- Remember, there are no silver bullets. No technological advances will relieve the software developer of the need to “think,” regardless of what the salesmen and trade papers tell you.

These ideals are essential whether you work for a software vendor or within the data processing department of a corporation. Once these hurdles have been jumped, developing quality systems should be a snap, right?

References

Yourdon, E., The Decline and Fall of the American Programmer, Yourdon Press, 1993.

Watts, Humphries, et al, Software Engineering Institute Capability Maturity Model, Carnegie Mellon University, Pittsburgh, PA, 1992.

---

[1]The percentages quoted have no basis in reality and are used for example only. The authors wish to stress their belief that both cars are outstanding products and have no desire to denigrate either.

From System Development, May 1995.

Home.