# Guerrilla SQA & Metrics: Conquering the Land a Bit at a Time While Leaving the Developers Alone

Scott P. Duncan
Quality/Process Improvement Consultant
SoftQual Consulting
3 Mink Court, Midland, GA  31820
(706) 565-9468
softqual@mindspring.com

## Abstract

Adequate resources and senior management commitment are oft-cited prerequisites to successful process/quality improvement activities. What if you have neither?  The metaphor of guerrilla fighters seems appropriate for the inch-by-inch “underground” effort which can characterize in-the-small process/quality change attempts. This paper suggests how to introduce software quality assurance (QA) and measurement practices where they might most easily be accepted without taking mainstream development tasks head-on.

The paper begins by describing an in-the-small, limited resource (“guerrilla”) approach and the context in which it is applied. QA and measurement practices are recommended which support one another and encourage data-centered decision-making. Implementation behaviors to avoid are noted to reduce the resistance faced when introducing such practices flow.  A connection is also made between these practices and project risk management since they complement one another.

[Note: this paper’s intention is to provide background to the two talks at this joint conference, allowing the presentations to focus as much as possible on examples of the “guerrilla” approach in actual project situations.]

## The “Guerrilla” Approach –

Guerrilla fighters, as opposed to a formally recruited and supported force, have to replace resources with dedicated belief that what they are doing is worth the sacrifice.  The armed force likely has the authority of “government” (corporate management) behind them while the guerrilla can, at best, hope for benign neglect from the formal hierarchy as long as they don’t seem to be in the way of “real work.”  The guerrillas often encounter resistance from the mainstream “ruling party” (development organizations) who, in all fairness, are given other priorities and whose leadership (middle management) has, indeed, been rewarded for succeeding with the behaviors now in place.

The guerrilla fighters learn to “live off the land” finding the things that feed people’s desire for change/improvement. Because resources are limited and because those they seek to help are usually less adventurously inclined, the guerrilla fighters’ “weapons” (methods and technology) have to be simple. These restrictions also mean the effort must achieve results through “quick strikes” (short-term) not lengthy “campaigns.” And to sustain the gains achieved, a “power to the people” philosophy has to be adopted since the guerrilla fighters, as they move on to the next “encounter,” will have to leave behind (technology transfer) the ability for the people to carry on themselves with few resources and basic technology.

Finally, guerrilla fighters come to understand that not all coups are successful and not all guerrilla leaders become heroes.

**Context for Introducing the “Guerrilla” Approach –**

The guerrilla approach is based on in-the-small activities which build up, bit by bit, over time into evidence of some “better way” of doing things. If you listen to people talk or read what they write about in-the-large success, one thing becomes clear: an essential element of the success was making it mandatory. One way or the other, management is given the message that being a part of the program is a requirement for staying with the organization (i.e., a “condition of employment,” as one senior manager put it in a process conference keynote talk last year). Even with such support, there will often be resistance from middle management who have “heard it all before” through various flavor-of-the-month programs and who have learned over the years how to weather the political tides in corporate existence to achieve what they perceive to be the “real work” expected of them.

The guerrilla fighter must be sure that, no matter the size of the organization, someone feels that things are not “fine as they are.” Dissatisfaction with the status quo which is identified and accepted above middle management will usually get the resources and support (or at least pressure applied) to make something happen with formal organizational visibility. The guerrilla fighter deals with the “local population” (line management and staff) who sense a problem but cannot get the attention and resources to attack it head-on.

Development groups produce the fundamental deliverables to make a software system “happen,” i.e., the executable code and related system elements. They often receive the largest share of project resources (e.g., people, funding, schedule) though they are regularly asked to do with fewer of them. They also usually have the most technology support for what they do because of years of industry focus on source/executable deliverables and the automation potential inherent in such deliverable formats. On the other hand, they are usually faced with the most unpredictable technical situations, constraints, and choices. All this inclines them to be reluctant to have “outsiders” tell them how they “should be doing things.”

Therefore, the guerrilla approach recommends a path which bypasses such “strongholds” in favor of working with areas in the company which may receive less attention, fewer resources, and little technology support: requirements management, project tracking, system (independent) testing, and client (hot-line) support. All of these areas are critical to how the customer is likely to perceive the end product, its support, and its evolution (i.e., viability over time). From a customer perspective, the assurance of software product quality is likely to rest with these organizations.

**Focusing Attention On SQA and Measurement –**

A large part of the software industry has come to equate QA almost exclusively with activities related to testing code and, upon hearing the word "metrics," envisions counting lines of code (LOC) or function points (FP). Given this view, the broader range of QA activities may either not be performed, performed consistently, well-coordinated, or monitored for effectiveness while metrics/measurement programs are seen as requiring development staff and management to take on data collection and reporting tasks with little return for the time invested.

Fundamentally, software QA and measurement are about achieving quality through greater visibility and more deliberate communication about how the customer's requirements are transformed, bit-by-bit, into a functioning software product. Despite the significance of the role of testing in the development of software, it is an after-the-fact QC function which identifies errors that already exist in the software product. Test personnel can have important QA impacts earlier in the lifecycle which will be discussed, but the activities related to testing code fit the QC function not a QA one. QA is foremost about trying to prevent, not detect, errors and uses detection (QC) results to that end. Measurement provides support for evolving a data-centered approach to focusing QA and improvement efforts.

Though it sounds strange on the face of it, "depersonalizing" quality is an important first step in reorienting thinking about software quality. By "depersonalizing" is meant, not removing personal responsibility for quality or for what is produced, but removing a too individual view of project deliverable ownership. If individual egos and personality become identified with specific deliverables, not overall results, it will be difficult to achieve the necessary visibility and improved communication for effective QA and measurement to grow. Unless there is a shared responsibility for the quality of the transformation from requirements through testing, problems may be viewed as purely matters of individual competence and performance, leading to a focus on the deliverer rather than the deliverable. Because of the culture of many organizations, the guerrilla approach of avoiding mainstream development activity is often the line of least resistance in beginning to make this point.

It will also be important to develop the understanding that QA is about more than testing code since this is what large portions of the industry, fueled by recruiting advertisements, define it to be, especially in commercial product development and in-house information technology environments. For this reason as well, focusing attention away from mainstream development helps to develop a broader view of what QA (both validation and verification) should mean to an organization in "building the right product right."

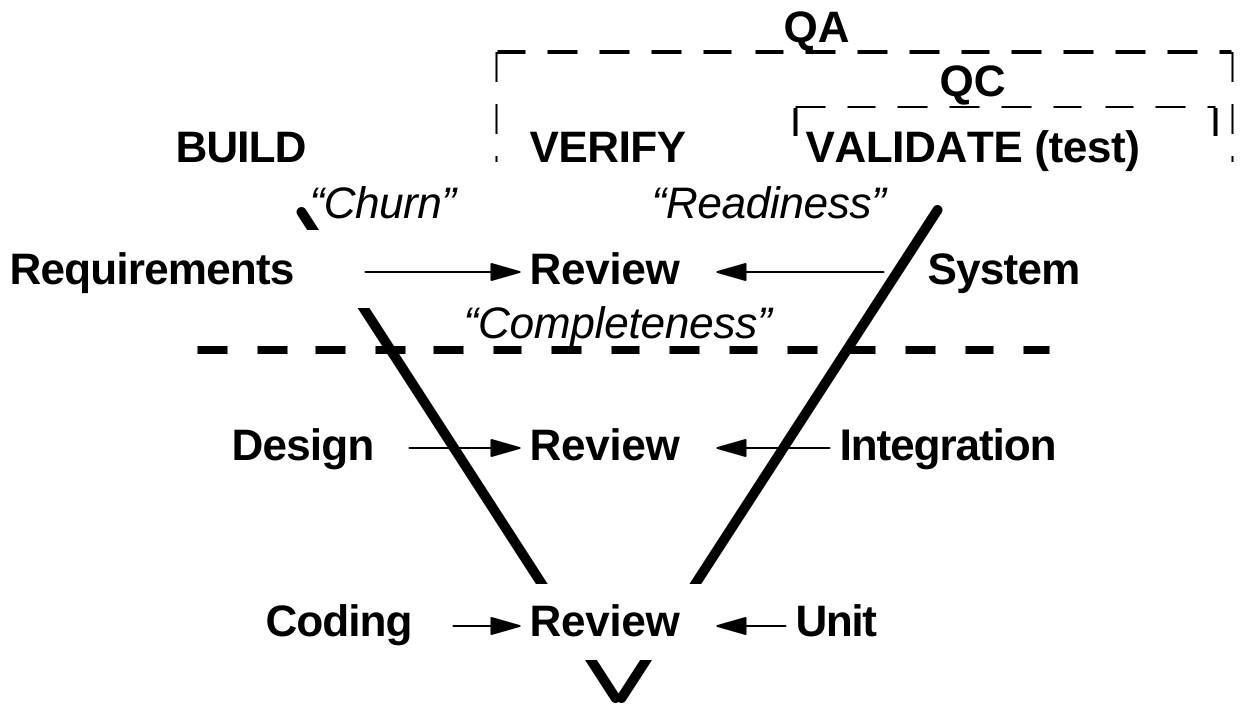
A broader view of QA should also be a way to encourage the development of a project culture based on data-centered opinions. The practice of collecting and using data to provide objective background for decision-making is constructively introduced through guerrilla QA.  Guerrilla QA includes data collection for immediate, local needs, which can become the basis for eventually providing information of use in the mainstream development lifecycle.

Finally, the link between QA, measurement, and risk analysis and management needs to be made. Effective QA and measurement contribute to risk management by providing feedback on how a project is progressing and where potential problems may be encountered.  Developing confidence in the predictive value of such feedback ultimately leads to a willingness to take on more aggressive project goals with greater confidence.  In return, risk analysis helps establish specific quality-related goals which provide the visible needs to which QA/measurement practices can be applied and by which broader application of practices can be justified.

**The V-Shaped Model –**

Making some of these points about software QA and measurement may become easier using a V-shaped lifecycle model of the software development process common in Europe over a decade ago.  Its main purpose at that time was to make the link between major software product deliverables and the validation (testing) activities associated with them. Adding a few elements to the original diagram makes the context of the guerrilla approach and its QA/measurement philosophy more clear.

The original model was the “V” with requirements, design, and coding positioned to represent higher to lower levels of specification which parallel unit, integration, and system testing representing internal logic, interface, and external functional validation of the product. Elements added to the model for the sake of this paper include labeling the Build-Verify-Validate aspects of the lifecycle, the quality assurance-control (QA-QC) scoping, the positioning of review processes, key measurement areas, and the dividing line between mainstream development tasks and those likely to be conducted “independently” of development organizations.



**The (enhanced) V-Shaped Model**

Starting at the top, the QA-QC scoping exists to emphasize that QA encompasses more than testing: it includes all verification and validation activities which means reviews and walkthroughs as well as forms of compliance/effectiveness assessment. If an organization does little or no assessment activity (i.e., post mortems, lessons learned, audits, process assessments) and handles reviews and walkthroughs relatively informally, then, ultimately, the only QA activities left do become those surrounding testing. However, in almost all other industries, direct inspection/testing of the product produced is considered to be a QC activity.

Below the QA-QC differentiation is the Build-Verify-Validate labeling which seeks to identify the distinction between creation of major product deliverables, verification that the outputs produced match the input specifications, and validation that the ultimate software product meets the specifications at each level. This distinction has been based on the IEEE Standard Glossary of Software Engineering Terminology (IEEE Std 610) in which verification is associated with phase-by-phase review of deliverables and validation, with testing whether the product satisfies initially stated requirements.

The review column has been added to indicate not only the verification activities, but also to show the link between basic product deliverables and validation planning and personnel. Those who are expected to validate (test) software products should be involved in a review of their specification at each appropriate level of granularity.

The model above has the measurements to be discussed later added to it to illustrate the "above the line" focus of the guerrilla approach. Requirements "churn," milestone "completeness," and system "readiness" for validation testing -- along with field "performance" from client (hot line) support records -- form the basis of measurement recommendations. (While client support is not shown in the model, it fits well with the external focus represented by requirements and functional testing.)

The requirements and system testing level of the model is the focus of the guerrilla approach since it is above the line separating it from the majority of internal development activities. As noted above, resistance to QA/measurement effort is likely to be stronger below this dividing line than above it. Thus, trying to introduce these activities above the line is initially almost always more welcomed.

**Behaviors to Avoid –**

It is quite common for in-the-large efforts to set up a committee of some sort and have them go off for a while and define standards, processes, procedures, and practices quite apart from the rest of the organization, presenting them, eventually, to senior management for approval so implementation can proceed with some authority behind it. Defining practice in isolation, of course, while encountering little initial resistance since you aren't "bothering anyone" much, usually ends up with substantial implementation resistance since there is no ownership of the resulting methodology by those who have to follow it. This is why the appeal to authority ("do it because so-and-so says to") is often necessary.

However, in-the-small guerrilla situations can suffer from a different form of isolation given that the people being helped often accept such help with the condition that they are not "bothered too much," as well. Thus, their reaction may be to suggest you "go off and put something together for us to look at" and not want to spend much time at all discussing with you up front what they need, i.e., short-cutting the formal requirements aspect of the effort. While understandable, it leads perhaps not to implementation resistance, but wasted time as you try to "invent" practice for people.

In the guerrilla approach one tries to overcome this by picking sufficiently small changes such that they can be presented almost completely as is (i.e., they are likely reusable from group to group as is with minor customization). This approach also avoids documenting practice too early. The basic practice is pre-documented since it is generic and any customization is documented only after experience with it convinces the users that it does, in fact, help them and they are willing to adopt it. Documenting local implementation is then less onerous and the users may, themselves, be willing to take it on.

Another area where in-the-large efforts often bog down is by collecting data that isn't used. There is general agreement that having "real data" is a good thing, but the data that committees recommend often includes much that is not currently captured. This requires substantial effort (usually at some corporate or senior management staff level) to design the methods and tools needed to capture the data. The result is often that use of data is delayed further, making the effort suspect from the perspective of those who are expected to ante data up to the corporate repository. Thus, the guerrilla approach seeks to use data that is almost assuredly already collected in one way or another or can be relatively painlessly collected. They key is to feed that data back using simple analysis or reformatting such that some "instant" information results.

Another measurement behavior to avoid is debating the appropriateness of LOC versus FP as measurements. The guerrilla approach makes this easy by having none of the measures it recommends based at all on using product sizing. There are pros and cons for both and, if pressed, FP would be preferred to LOC -- though there are certainly acceptable ways to use LOC -- but product sizing does not need to be addressed when you are just starting out, especially for in-the-small efforts. (A product sizing view, even with FPs as the measure, usually results in a focus on development deliverables which the guerrilla approach tries to avoid.)

Finally, in-the-large QA/measurement practice eventually gets around to determining if people are properly complying with new methods and practices. While compliance checking is a common QA approach, putting compliance before effectiveness has been assessed cannot be recommended. The guerrilla approach emphasizes effectiveness as the basis for assessing any aspect of software development products or processes. One can easily ask the question of whether or not a given product or process effectively results in achieving the goals (requirements) defined for that product or process. This has the benefit of emphasizing that there should be such requirements, making sure they are accepted by everyone involved (stakeholders), and focusing on what makes the results effective rather than simply compliant to dotted "i" and crossed "t" norms.

**Practices to Advocate –**

Because the guerrilla approach focuses "above the line," source and executable code as well as design deliverables are not the major concern. The focus is on those who create requirements, have to validate them, need to support users who wanted them, and seek to keep their realization on track. Guerrilla QA and measurement treats mainstream development as a black box, asking only that development groups look for clarification of any and all requirements questions and design decisions affecting what an end-user will see or experience when the specifications they receive do not make this clear.

A couple famous movies have characters who tell us to "Show me the money!" and "Follow the money." Guerrilla QA replaces "money" with "requirements," following them through the lifecycle to verify their transformation along the way and validate their ultimate realization in an executable system. Therefore, traceability, and anything associated with making traceability happen, become main guerrilla practices. The path from requirements creation, through their review and change management, to their ultimate validation is the "money trail" to follow, using a few measurements to see how smoothly and predictably that flow is progressing.

Traditionally, traceability means the ability to identify individual requirements and follow them through the lifecycle to be able to demonstrate that none are "lost" along the way, i.e., that every one specified gets incorporated into some element of the design and is properly validated at the end. However, to do all this involves not just an identification scheme for individual requirements, but a complete process including stakeholder identification, review, and change management. With the one caveat above, regarding communication of ambiguity, this can all be initiated with little mainstream developer involvement – "little" because some level of development organization involvement in requirements review is part of initially trying to weed out ambiguity and give development organizations as early a chance as possible to understand what is being asked of them. Appropriate review also spreads responsibility for the deliverable to the entire team even though it is an individual or identifiable group of individuals who create the deliverable.

This first step is to be sure all stakeholders in the requirements are given a chance to participate as soon as possible in the review/comment process. In particular, testing organizations should be involved early to determine whether, as stated, requirements can be tested and with what level of difficulty/complexity. The role of testing organizations in requirements review cannot be underestimated. Not only can it ensure that requirements have meaningful validation applied to the resulting software, but this clarification makes it easier for development to do their job with less ambiguity and, therefore, less likelihood that unstated requirements “interpretations” will make it into the design and implemented software. Consideration of the “testability” of requirements can go a long way in uncovering customer expectations that will be difficult to satisfy as well as identifying possible risks in their implementation. For a similar reason, client support areas should also be included in initial requirements review because of their perspective on how customers are likely to expect the software to respond to their requirements given prior experience with customer feedback on the product.

Inevitably, though, requirements seem to go through various iterations as more is learned about the nature of the system the customer wants. Various requirements elicitation techniques and prototyping methods are used to make it more clear to the software supplier just what the customer would like to have. But no matter what techniques and methods are used throughout this process, ensuring adequate change management is an important QA practice to encourage. The measurement recommended for this is to record the number of times a given requirement changes, both before and after baselining, using nothing more elaborate than a spreadsheet with a column for the requirements identifier and a column for the count. The result, nether good nor bad in and of itself, will be to highlight where change is occurring which can signal areas where there is less clarity. The reverse may also be true, i.e., no change to an original requirement might suggest it be reviewed in light of related requirements that are changing.

Though not often looked upon as a QA concern, milestone/resource tracking provide insight into whether anticipated efforts are progressing as expected. If they are not, asking why not usually reveals potential quality risks. It is so common for projects to complain about tight schedules and for in-process milestones to slip that talking about it just becomes part of the general “noise” in software efforts. Applying some simple measurement and analysis to milestone data can be revealing when the percentage of missed milestones and average days missed are calculated and examined. Again, this can be done with a simple spreadsheet of 5 columns (original date, (current) adjusted date, actual date, days overdue from original date, and days overdue from adjusted date) plus a percentage of overdue versus total milestones and an average of days overdue. Such data can turn “noise” into something that gets attention as the “completeness” profile of a project is developed and the “90% done” syndrome is tracked.

If we view development as largely a black box for the sake of the guerrilla approach, then the “readiness” (acceptance criteria) to enter than box from requirements definition and to exit it into system (functional) test become QA focal points. Naturally, both of these can be done incrementally given that acceptable subsets of requirements and testing plans/scripts can be designed. But whether readiness is assessed on a whole system, or a subset of functionality, the approach is the same: what is the criteria for passing through this “gate”? This question may make it appear that an in-the-small approach has adopted in-the-large assumptions about affecting development, especially with the readiness to enter testing.

Those who define (and review) requirements can set their own “readiness” criteria and the rate of change (“churn” measurement) can be one such criteria. If some significant percentage of requirements (or critical requirements) continue to undergo change, it may not be possible to “stop the production line.” But it can be a sign that a risk is being taken and that active monitoring of continuing change is required. At the other end, as the system exits mainstream development and is presented to independent testing, certain “sanity” testing with pass/fail thresholds should exist. Again, it may not be possible to “stop the production line” -- especially as a committed ship date comes closer and closer. But system test can, at least, identify the readiness using such criteria and track that against problems found during testing and the number of rework/retest cycles to which such readiness compares. Such data may later be used to suggest the “cost” incurred through a lack of readiness combined with the demand to meet dates regardless of that readiness.

Once a project is completed, it is difficult, if not impossible, to get formal post mortem or lessons learned feedback on the effectiveness (or lack) of the development effort. Everyone is off doing the next project(s). People have forgotten many details (or hope they can). And unless a serious problem occurred for which accountability must be identified, the feedback will not happen. The guerrilla approach suggests two things: in-process 1-minute lessons learned and the half-hour historical data collection.

The former involves getting in touch with key people from requirements, system test, project tracking, and client support every other week or so to ask “What’s the key issue you’re facing?” This usually only has to take about a minute, but once engaged, many folks will gladly offer more. The historical data collection involves going to people soon after the system is turned over to production and collecting some basic data about what went on in the project. Some of this data may be available from automated sources since it involves length of time and effort in various phases of the project, but the overall goal is to engage people in discussion and help them be more reflective about what happened. The goal is to do this without making it seem like some major meeting has to be arranged and to encourage more candor than might be comfortable in a room full of peers and management. It is usually fascinating what such one-on-one interviews produce. Both of these efforts trade the guerrilla fighter’s time for information since they are structured to be of reasonably low impact on the project members while they may, collectively, take hours of the guerrilla’s time.

Finally, it is important to gain some sense of a system’s “performance” in the field. Quite often this means some defects per thousand lines of code (KLOC) measurement, etc. The guerrilla approach uses data from client (hotline) support (which is where the defect reports usually originate), but it tries to use all the data to form a picture of the impact of the system in the field rather than isolating individual defects. The guerrilla approach advocates using client-days in the field (number of clients where the system is installed times the number of days its been installed in each one) and number of failure reports collected as the key measures. The former avoids the need to compute size measures while focusing on a measure that reflects a client perspective on use of the system. The latter reflects “faults” (each time the system is reported to have failed) rather than “defects” (specific instances of a system implementation error). Faults are what customers encounter; defects are what developers find and fix. Defects may be getting recorded and tracked since it is likely the most common measure any software organization makes. But the ”above the line” and in-the-small guerrilla approach focuses on faults and the hotline data since there is generally less resistance about getting the data and less controversy over how to classify it.

**Eventually Including Development** –

As noted earlier, eventually the goal is to expand broader QA and measurement practices into mainstream development activities. However, to do so while maintaining the “guerrilla” approach, i.e., in-the-small with no organizational pressure and “programs” forcing the issue, means doing three things:

- getting the attention of development groups (by offering insight into what they are doing),
- highlighting development performance (by providing guidance to focus improvement), and
- supporting development quality improvement (by helping implement improved QA and measurement practice).

Getting attention is not hard when it is done on a peer-to-peer basis rather than through organizational edict. This is exactly how the guerrilla approach operates, encouraging only enough “visibility” to allow those closest to the issues to take action. Providing data to people and asking them what they think it is telling them about their work as well as offering to assist in addressing any problems the data reveals yields surprising results on the peer-to-peer level. Testing organizations are usually well-positioned to provide such information though they are often not consulted in this way since their function is largely viewed as “bug-finding.” Peer-to-peer interactions outside of formal status meetings help overcome this stereotypical view and increase the respect testing receives from development groups.

**Conclusion –**

An in-the-large QA/measurement program often starts with hierarchical reporting and tracking which leads to a “give them what they want to hear” attitude. In this way it often co-opts QA and measurement before they can get started at the line staff and management level.

The guerrilla approach is based on in-the-small, peer-to-peer efforts to raise awareness regarding broad QA and measurement concerns. It attempts to avoid major sources of resistance by not injecting itself into mainstream development tasks. It seeks other parts of the organization where data is already collected and an interest in improving the process through greater visibility and communication usually already exists.

The ultimate goal is to be able to offer such information to development organizations, showing them how it might be used to highlight issues they are facing and leading them to changes which increase product quality without inordinate effort on their part.

# Scott Duncan

Scott Duncan has 27 years of software development and quality consulting experience in publishing, law and public safety, agricultural R&D, commercial software, finance, and telecom.  The last seven of these he has worked with line staff and management, from organizations of 20 to 600 people at a time, to meet quality and improvement goals, including lifecycle-based software quality assurance and measurement/estimation and ISO 9001 (TickIT) registration and CMM Level 3 assessment.

Mr. Duncan participates in ISO/IEC standards work; is standards chair for ASQ's Software Division; has been a speaker, seminar leader, and program committee officer at national and international user groups and conferences; has spoken and/or taught at colleges, universities, and companies in the United States, Canada, and Europe; and has helped organize and serve in various officer roles in local chapters of the ACM, AITP (DPMA), ASQ, and IEEE CS.

SoftQual Consulting
3 Mink Court, Midland, Georgia 31820
706-565-9468
softqual@mindspring.com