# PROCESS DISCIPLINE IN THE INFORMATION AGE: RETHINK THE QUALITY ABSTRACTION

Hillel Glazer, Principal & CEO, Entinex, Inc., 1516 Castle Cliff Place, Silver Spring MD 20904, USA, +01.301.384.4203, hillel@entinex.com, www.entinex.com

## Original Abstract

"Process Discipline in the Information Age: Rethink the Quality Abstraction" acknowledges that software has never been more critical to, nor as widespread as it has become in the "Information Age". The demand for software, both in terms of who wants it and how much of it they want has grown logarithmically. As more of the business world relies on software, increasingly the speed of software development has become critical to business success.

Some of the demand for speed has been sated by the ease and flexibility of web-based and object-oriented software development and deployment. Unfortunately, this ability has become a blessing and a curse to the entire software industry. The ability to develop quickly and make changes on the fly has become the customers' expectations. In response to the pressures of the information age, developers created lightweight development methods that many consider to lack the disciplined processes critical to producing quality software products.

The argument from advocates of process discipline is that the lightweight methods rely heavily on testing and on unwritten processes, requirements, and commitments. While these observations may be true of some so-called lightweight developers, it is not true of the overall concept of lightweight development. In fact, what may be keeping process disciplinarians from considering or implementing lightweight methods may have more to do with the legacy of process discipline implementation rather than anything to do with lightweight development.

The challenge remains: how to develop quickly enough for the demands of the information age and still maintain a level of process discipline appropriate to achieve organizational needs. Developers and managers have been grappling with this challenge long before the information age became a household term. (Originally, however, the schedule crunch wasn't due to market forces, but to poor project estimation.) Nonetheless, today there's a very real need to satisfy the customers' demand while simultaneously addressing the development companies' control of cost and resources.

This discussion will address how "Rethinking the Quality Abstraction" can help companies achieve process discipline without impeding development speed. it will look at Extreme Programming (XP), a popular lightweight method, and at ways of conducting Quality Assurance within the XP environment, thus achieving the process discipline sought by many organizations as necessary to control costs and predictably produce quality products.

## Introduction

Rethinking the "Quality Abstraction" must happen if software quality assurance (SQA) practitioners are to move SQA practices forward to align with today's development environment, market forces, and technologies. This paper uses "Lightweight"/ "Agile" software development merely as a vehicle used demonstrate the need while also providing a brief introduction to a widely misunderstood development method.

SQA suffers from a legacy. This legacy is a mindset impressed by decades of development for large, complex projects in which software was merely a component of the overall product, coupled tightly (technologically) to that product whose role was a more efficient way to perform similar logic using electronics. This mindset has since lost its validity as software has become the entire product, no longer tightly coupled with the hardware it will run on.

Fundamentally, software products in today's information age are dramatically different from software in the Cold War era. And, so too is software management and development. The issue is: software quality assurance has not appreciably changed commensurate with the technologies and methodologies. The majority of today's SQA instantiations still reflect Cold War era QA techniques that simply don't work in today's development environments, for the demands of today's software markets, or the expectations of today's software companies.

Evidence of this can be found in the statistics of companies who achieve and then abandon their software process improvement efforts such as CMM/CMMI®. Supporting evidence is gleaned from discussion and user group conversations among developers. The movement among software developers towards more agile, less bureaucratic development and management methods cannot be ignored. These methods are often referred to as "lightweight" practices in contrast to what is perceived as "heavyweight" (heavy-handed) practices found in many government-sponsored approaches.

Where "lightweight" and "heavyweight" methodologies collide has more to do with attitude than actual strengths or weaknesses of either approach. The legacy approach to SQA bred QA programs that turn out to be very similar from standard to standard and place to place. This, in turn, breeds a way of thinking about lightweight software development. The thinking goes something similar to: *there's no such thing as robust QA in lightweight development*.

This paper poses the thesis: if QA practitioners view lightweight development as undisciplined, perhaps it's the way in which QA has been historically applied that makes this so, and not necessarily anything inherent in agile method. A more realistic middle-ground is sought in which lightweight/agile development can also be robust. It is found by seeing how to make QA appropriately agile as well. The premise being: were the mindset about QA to be "re-set" by changing the abstraction, the new abstraction could then be applied to QA for any environment and could even be used to improve current QA implementation in non-lightweight software environments.

One assumption made in the paper is that QA activities are expected to be a value-added component of a comprehensive product development process. By looking at QA in terms of its basic goals and how to adapt what QA professionals do to meet those goals, a QA approach can be developed that works in any environment and still be in complete compliance with standards, and policies. A change in abstraction will cause how QA "shows up" on a project, not what QA is expected to accomplish.

## Understanding "Lightweight" / "Agile" Development

The actual characteristics of lightweight development vary widely from organization to organization. Lightweight development bears the reputation as undisciplined, often due to narrow implementations of the concepts, rarely following any formal development guidelines. Thus the reputation is unfair. Developers who code without rules, process discipline, or many other management tools ***are*** undisciplined. This is ***not*** what agile development is, any more than the original intent of effective QA was to be heavy-handed.

The purpose of lightweight development is to allow for better productivity. The enemy of productivity is heavy-handed process controls. Admittedly, some developers pursue lightweight development hoping to shed controls, checks, and balances necessary to make good products. This is far from what lightweight is about.

If this were true, then lightweight developers would be operating under a *modus operandi* that reads, *"produce quality software in the absence of any process"*. This would be absurd. Even the most ardent supporters of lightweight development would not agree that this is what takes place. It's not the absence of process that makes a development method lightweight, it's the absence of *unnecessary* or *obstructive* processes that makes a method lightweight. With this in mind, a working definition of "lightweight" could be:

> ***The minimum, most unobtrusive approach to developing software that produces a quality product when the customer expects to get it, at the price they expect to pay.***

Alistair Cockburn, widely acknowledged as one of the "fathers of lightweight" having developed the "Crystal" series of development methodologies writes,

> "Computers must support the way in which people naturally and comfortably work. This is needed both for personal job satisfaction and for corporate survival. I care about whether the team is thriving, and whether the software is coming out the door. Keeping the people trained and the process light are key to both."

Someone also attributed the following definition to Alistair Cockburn:

> "[Software development] methodologies that focus on the use of as little process as possible to obtain good results. Often proportioned to the size or risk of the project."

If among developers, "lightweight" were synonymous with "undisciplined" then why would Martin Fowler, another of the pre-eminent thinkers in all of programming, have taken so much paid to specifically include the role of process in his paper, *The New Methodology*? The themes that emerge from his outline are: business, progress & productivity, people, and processes. A group of similar-minded programmer-gurus got together and called themselves the "Agile Alliance". They formed a statement of principles which very clearly communicates a balance between process and progress.

Such statements clearly articulate exactly the role and value of processes. However their statement of principles is overshadowed by their more notorious (and poorly named) *manifesto*:

> *"We value:*
>
> *Individuals and interactions over processes and tools*
> *Working software over comprehensive documentation*
> *Customer collaboration over contract negotiation*
> *Responding to change over following a plan*
>
> *That is, while there is value in the items on the right, we value the items on the left more."*

The Agile Alliance isn't against processes. Process-oriented professionals nonetheless read this statement, and have found their "proof" that lightweight development is anti-process. Such interpretations come from people who have an established prejudice against agile development. Again using *reduction ad absurdum*, can anyone prefer the process to actually delivering the product?

## The Role and Goal of "Quality Assurance"

Realistically, in the typical QA approach, not all activities performed to "satisfy QA requirements" are productive, pro-active, value-added contributions to producing the product. Otherwise, developers would be open to using or allowing entry of a QA process. Any process that requires time away from development for documentation of work already performed isn't productive for the developers. The path towards reconciling heavyweight and lightweight practices will be found by bridging such gaps.

A working definition of Quality Assurance can be,

> *A process/effort that ensures that processes are followed, that the processes have us doing the right things, the right way, and when they fail to be used or fail to perform as expected we have a way to correct, adjust, or escalate the matter until it is resolved to everyone's satisfaction.*

From this, a policy statement can be made that reads:

> *All processes must actively support development's productive activities. Process must avoid creating additional effort for development functions outside of the project's stated development function processes. To ensure that processes are acceptable to the development functions, they must be designed in collaboration with the project's development community. The development community, in turn, must allow the process owners to achieve their process and product oriented objectives. Thereby reaching consensus on a balance between process and productivity. The goal is to fully integrate necessary process steps into activities that add value to the development effort while resulting in insight, predictability, measurements and traceability of process effectiveness.*

Though this may be wordy, it's a philosophy that must pervade all process design. Software, hardware, lightweight, heavyweight, whatever.

## The Historical Role of Quality Assurance.

QA undergoes continuous improvement in terms of its application as well as acceptance. QA is an official component of many project plans and a valued resource in many projects and organizations. QA can hold up a project with process problems, and "dress down" a project manager for skipping steps.

Unfortunately, QA is still sidelined too often when business needs take priority. Especially when QA has the reputation of "policing" rather than a contributing to the effort. Frequently, business owners will bypass managers and go directly to developers when such layers are seen as getting in the way.

Showing the business value of QA through analyses, 6-sigma SPC, and other techniques are still more *re*active than *pro*-active. What developers (and executives) want are processes that implement QA so that they don't slow progress, don't break momentum, and don't install the sense that people are being policed. Such processes are demoralizing.

Processes are fueled by people, and people hate heavy-handed processes. Developers seek "lightweight" methods in hopes of finding refuge from heavy-handed processes, and in their escape, throw out the mantle of all processes. The origins of QA standards explains much.

Early software projects were big, slow, and geographically dispersed with layers of bureaucracy designed around project management methods that also built tanks, planes, and ships. Based in the ability to exploit manufacturing work-flow controls, when applied to software development these QA methods fail to achieve their intended goals. Software developers have long known that the software development paradigm shares very little with the manufacturing paradigm, but the methods of performing SQA have not made the shift across the industry.

Defense and similar large-scale old-style projects shaped much of what is known today about QA. Compared to today's technologies and the speed with which software reaches the market, such legacy projects provide a very limiting pool of experience.

When ISO 9000 came out, many organizations following Mil-Q-9858A made a transformation matrix to convert Mil-Q into ISO 9000. When CMM® came out, ISO 9000 and DoD-STD-2168 followers made a matrix to transform ISO 9000 activities into CMM®. Whether government, commercial, or home-grown, with each introduction, too many organizations simply took the old ways and made them fit the new requirements. Many QA plans now look the same from one organization to another because their foundations all spring from the same basic set of requirements that have hardly changed in decades! Are so many businesses and projects so similar that they can all use that same QA approach? Of course not!

## QA in Business

The non-productive activities and paperwork created by legacy QA abstractions is most felt at the development level. In many large, complex projects, the additional effort and time needed to follow the processes are easily absorbed by the project. The pace of these projects are such that the deliberate (if not judicious) addition of time and work can be handled.

Development needs to stay productive, control costs, and keep people motivated. The pace of the project should not be overshadowed by the effort to follow the process. Lightweight development recognizes the need for processes that allow a project to get done at the pace of the project. Many processes, QA included, have fallen short because they do not account for the pace and complexity of the project.

Modern software organizations and projects need QA processes that more closely fit each project, dynamically adapt to the project and make development cheaper, better, and faster on every subsequent project. These are the attributes developers, project managers, and business owners are looking for. These are the attributes that truly add business value to the QA process. In the commercial market-driven world, on time working product is a must. Processes must reflect the demands of the customer. First and foremost. Processes must be adaptive and scalable to handle exceptions.

Legacy QA processes, historically, have been designed without attention to business goals. Even while models like the CMM® enjoin users to ensure processes are adding value to their efforts few implementations ever achieve that ideal. Instead of investing in improved processes, too many companies believe the easier route is to supplement their existing processes with disruptive, paper-intensive meta-layer of activities that produce evidence that a process is being followed but do not contribute to productivity. That has not changed in decades.

## A Word About Development Processes

There is distinction and noteworthy interaction between development processes and management processes. Hardware, for example, can be designed and manufactured in any one of several ways. The design can be on paper, or using Computer Aided Design (CAD) systems. Manufacturing can be by skilled artisans or can employ automated systems. These are the "development methodologies". Tools and tool control, inspection, inventory control, materials ordering, environmental controls, organizational needs, and so on also characterize the manufacturing environment. These are "management methodologies".

The development and management methodologies, therefore, are distinct. While not completely de-coupled, one does not dictate the other though they must complement and support one another. They work together to achieve business goals. It is desirable that they are optimized to work in the same business and operations

strategy models. Fundamentally however, whether blueprints are drawn by hand or by CAD is not dictated by how the flow of material is controlled through the plant.

In the software world, for example, CMMI® doesn't care what development methodology is used. It doesn't dictate use of the "Waterfall" model, or that XP is better than "Crystal Light", and so on. Distinguishing the software development methodology from the software management methodology eliminates one of the barriers to managing QA in lightweight development environments.

## QA in the Context of Development Processes

QA's role in the development process is fairly simple. No one questions that development standards, methods and processes need to be followed and need to work well for the project.

QA is responsible for ensuring that the project's methodologies are taught to new developers on the project; ensuring that the methods are followed by everyone, and ensuring that QA activities for projects are planned and not spontaneous. QA must measure the effectiveness of the methods, provide visibility to management via appropriate metrics from prior project QA experience, and know when the methods need to be adjusted. A person independent of the political and organizational chain-of-command are necessary to avoid conflicts-of-interest to achieve appropriate objectiveness from the product and its stakeholders.

QA must support the project in achieving the intended benefits of the standards the project sets for itself. If the project's processes and activities do not promote or support its standards, policies, or methods, it's QA's job to bring this disconnect to the attention of the people who can make appropriate changes.

For the most part, Software QA boils down to making sure that the things that need to take place can happen and are happening, and that when they don't they get fixed. Everything else is technique.

"Keeping things simple" is critical to a well-formed abstraction. When QA is distilled to the above statements, possibilities are created regarding how to look at the organization's QA processes so that they can operate in any environment.

Lightweight development doesn't mean there are no requirements management, or no QC, or no QA, or no CM, or project planning, or project tracking, or reviewing of designs and work. Development without those things would be called stupid programming, not lightweight programming.

## Rethinking the Quality Abstraction

Many QA processes rely on generating artifacts, evidence, and other labor-intensive "bread crumbs" tangential to the work being done on the product itself. These tangential efforts rely on the same people as development and therefore cannot occur in parallel with the production, thereby increasing the amount of time it takes to carry out a project.

This approach to the QA process not only relegates QA to the role of policing and gate-keeping, but drastically minimizes the positive impact of the overall effectiveness of the QA program. One can seriously (and not without merit) question the timeliness, contribution, and overall value of QA when the activities defined by or for QA purposes do not benefit the project.

"Proactive" QA is still seldom proactive throughout the lifecycle of development. Proactive often means a level of effort before a project starts, followed by an unsteady tempo of reactive activities that are only conducted as events unfold. It's this entire approach that needs "re-thinking."

Instead of this approach, the software industry needs QA process that ensure processes are matched to project objectives before the project gets under way. Processes that get into the detail of the standards and methods so that when the standards are followed they automatically generate the necessary "proof" of process compliance. Not processes that simply create automated markers and flags, rather approaches that push metrics and data generation into the process so that the successful output of the process is only possible if the process was properly followed.

Using the technology of the development process, tools, and standards as the medium to collect process and tracking data would accomplish much of this. Instead of policing the processes through post-mortem artifacts, QA could be free to analyze the effectiveness of processes in real time and make adjustments.

The actual abstraction transformation is simple. Instead of focusing the quality process on the effort of proving a formal process is followed, ensure that the processes are effective, productive, and valuable to the goals of the business, and create production methods that produce the evidence as a by-product of the effort rather than a separate activity.

Transforming the QA abstraction from an investigative approach into a productive, business-driven, value-focused umbrella of activities that improve the development effort, will achieve the "rethinking" of the QA abstraction that is necessary for lightweight development methods.

## Examples from Everyday Activities

A Configuration Management tool can tell whether the CM process is being followed, or whether certain other activities have taken place on a particular item. Development teams can create their own coding and commenting standards that allow for ease of assuring and assessing whether those standards have been met, or even to simplify whether maintainability requirements are met.

Naming conventions, user authentication rules/policies, and code release processes can be created that only move code forward under certain conditions. Time-sensitive configuration states can be assigned that indicate build includes with authorizations and test results. This can be rolled into a process that allows independent but automated review of the deployment status for use with project tracking.

Requirements management process/tool can also be connected to the risk management process/tool which can both be monitored by QA via their involvement with the mitigation planning and execution of special risk-reduction development activities. QA's involvement in such activities may require retraining for QA and other personnel. In other organizations, much of this effort can be pushed to the developers – without causing them more work.

QA's real job is to ensure certain information is generated to assure that processes are followed. There may be no need for QA to get mired in the specifics of development. By collaborating with developers on producing what QA needs, the everyday hour to hour activities of development can become part of development activities and QA can be left to monitor the overall effectiveness of the project's processes and feed back process improvements.

In a well integrated project, generating the data QA needs would merely be a report that runs every so often querying certain tables and build repositories. A QA program at this level of abstraction is infinitely scalable to any project as long as there's the will to cooperate for the purposes of benefiting the business.

## Conclusion

Many organizations' QA abstractions are a result of a long history of how QA has been applied. Understanding this history and placing the context of the role of QA makes it possible to see why there is mutual distaste between lightweight developers and process discipline practitioners. If QA practitioners worked to create processes that promoted productivity and if developers followed their lightweight methods diligently, then the two philosophies can find mutually beneficial grounds upon which to develop excellent software of very good value.

## Bibliography


- Beck, Kent. *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 2001.
- Glazer, Hillel. "Dispelling the Process Myth," *CrossTalk*, November 2001, Vol. 14 No.11, pp. 27-30.
- http://alistair.cockburn.us
- http://c2.com/cgi/wiki?XpAndTheCmm
- http://c2.com/cgi/wiki?CategoryPattern
- http://c2.com/cgi/wiki?QaIsNotQc
- http://www.extremeprogramming.org
- http://www.martinfowler.com/articles/newMethodology.html
- http://www.sei.cmu.edu
- CMM® and CMMI® are Registered Trademarks of the Software Engineering Institute.