V - 3: Focussed Quality Assurance

The purpose of Focussed Quality Assurance (FQA) is to rapidly put in place a simplified and cost effective Software Quality Assurance (SQA) system that is clearly aligned to organizational need. When it had demonstrated its value further investment can be made to develop it to become a full SQA system.

What it is for:	Provides visibility of working practices that satisfy organizational policies
When to introduce:	1. When a quality assurance capability is to be developed. 2. When policies for software development are required
When to use:	Conduct periodic audits of project's conformance to organizational policies.
When not to use:	When a mature and effective SQA function is already in place.

Software Quality Assurance is recognized in software engineering (but no necessarily in other software development cultures) as an important tool for ensuring the quality of software by providing visibility of development activity and ensuring good control of software quality. In organizations where SQA is already in place and its value understood it can be used by software process engineers, as well as software managers, to give visibility of the impact of changes to development and management practice.

In organizations without SQA establishing it can be a time consuming and problematic as its scope and function are developed. Problems often arise because a traditional, engineering type SQA capability may be inappropriate, not understood, or not valued. There are good reasons for this. Quality assurance originated in engineering and manufacturing organizations producing products that needed to demonstrate good intrinsic quality characteristics when in use. It is used to increase confidence that the products will have these quality characteristics. Where software is produced as part of engineering systems or products SQA will be found in place fulfilling the same function. However the majority of software developed is not for software products and is not required to have the same quality characteristics. Most software is developed as part of organizational infrastructure; e.g. payroll, order or other transaction processing, and increasingly, customer facing applications. This software is not valued as a *product*, but as a *service*. The level of service can vary widely and investment in the service level will vary accordingly. The essential difference between software as product and software as service is that when the software is used to deliver a service it is *part of a system that includes the people that maintain the system* and may have developed it too. The quality of service is determined by the performance of this human element

too, not solely the intrinsic quality of the software, as the quality of software products is. Because of this fundamental distinction the approach to quality assurance differs. If product oriented assurance is introduced it can overload development and support activities with administration, controls and other tasks that can be unhelpful, have limited value improving service delivery, and is quickly perceived as such..

To put in place an appropriate software quality assurance function, attuned to the needs of the organization Focussed Quality Assurance has been developed. The characteristics of FQA are; speed of implementation, simplicity, focus on business needs (hence the name). It is intended as an aid to newly appointed quality assurance managers responsible for putting in place an SQA system.

The usual, high risk approach is something like:

> *A Quality Assurance Manager is appointed. They are given responsibility for the planning and development of the SQA function. With no standards in place to assure the manager, drawing on previous experience or reference materials, drafts process standards, perhaps with some document templates to be completed and maybe some software quality criteria. A schedule of QA reviews and audit is planned together with the paraphernalia of a mature SQA system. (This can be time consuming, especially if senior management wish to review and revise SQA materials.) When all is ready a series of awareness seminars or tutorials is delivered and the reviews begin - often with some success as QA reviews identify development issues and project risks and SQA help developers conform to the new standards. As time passes the value of the SQA standards is called into question as little improvement in project performance or product quality shows itself and the SQA function becomes a routine irritant. The requirement to adhere to standards is challenged; rework to make products or processes conform lapses. Where dissension between the new SQA function (an overhead) and development and support projects (revenue generators or service deliverers) occurs the projects usually prevail, even though the quality assurance manager knows they are 'right 'and it is their job to defend the standards they have spend time and effort developing. The SQA function is undermined and its decline begins; the organization has rejected SQA.*

The initial success can be seen as a Hawthorne effect, together with the reporting of well known problems by an independent third party. The standards will have had little to do with the actual performance of work or quality of software, although they will describe some well recognized best practices. The senior management, after receiving SQA reports with some interest and enthusiasm, can fail to appreciate that the SQA function is part of a system within which they too are required to act.

The defining characteristics of FQA are speed of implementation, relevance to the organization and, initially, simplicity. It provides visibility to senior managers of software development activity and software quality, but only those aspects that are of interest to them.

To put FQA in place:

1. Senior management decide what they want to know about development and quality. This is described in a series of policy statements[1]. (Policy statement are often required by quality standards and process models, but often no more than token statements about excellence, world-class performance, etc. that rapidly become invisible to those in the organization.) FQA requires that policy statements are reasonable, relevant, widely published and capable of *binary audit*. Binary audit means that the aspect of software development or software quality addressed by the statement can be assessed and answered by a simple yes/no answer. (It is desirable to design statements such that 'no' indicates non-conformance, in much the same way as checklists are designed.) For example, a policy statement may state 'Each project will have a reasonable, up to date project plan.' The drafting of these statements can be used to get senior management to review and make explicit and specific statements of what they value and what they want. This has considerable value in itself; implicit values may be interpreted wrongly with major consequences: A real example - developers were working to 'delight the customer' by delivering all the functionality required, together with additional desirable features. The senior manager commented to external consultants that he wanted developers to meet the requirements. This they appeared to be doing. On enquiry he expanded his statement 'meet the requirement, *but no more*.' Additional features cost time and effort to develop and support, were not included in the contract, and this came directly from the company's profits. This had not been made clear to developers. A brief, specific policy statement could have made this explicit and unambiguous. Senior managers can have difficulty articulating what the business wants from the software development capability (hence the dependence on models such as the CMM). Draft policies, prepared by others, may be required to seed the policy-making activity as senior managers consider what they want. Starting with a blank sheet of paper can be difficult. The Good Software Practice (GSP) draft policies can be used for the initial selection. It takes time and skill to draft and agree a good, auditable set but should take no more than four to six weeks, elapsed time. A set of policies will be simple, clear and not take more than a few sides of

[1] The phrase 'policy statement' usually makes a manager's heart sink. They are seen as unfocussed , generic truisms or management gobbledegook of little value. FQA makes them effective management tools but it may be worth finding another phase to replace the term 'policy statement'.

A4. It may be useful to organize policies by into categories to make them easier to manage; for example project management, requirements, technical assurance, defect levels, training, lifecycle management... etc. The CMM's KPAs (each of which, in passing, requires a set of policy statements) are a good starting point. But remember that policies should only be selected or drafted if they are of interest to senior managers. Overloading the set of policies with worthy but uninteresting policies will discourage senior managers from examining the information delivered about them. FQA is intended to provide senior managers with information about development practice of interest *to them.* Policies can be elaborated or added to later.

2. Publish Policies. The policies drafted by senior management and are tested by review by those they will affect. (In some organizations the policies will simply reflect the established, consensus view of good practice and good quality. In other they may reflect changing business requirements. Whatever the situation this should be articulated by senior management.) The policies should be reasonable. Where senior management attempt to impose unreasonable policies, either by design or ignorance, the policies will be discredited and 'work-arounds' rapidly appear. User review is a valuable, low risk sanity check. Constraints, rationale, mission statements and exceptions can also be published with the policy statements. The publication of the policies can be used by senior management as an opportunity set direction, revive a quality initiative or express its collective will.[2]

3. Audit practice using the policies. With the policies published and agreed the new SQA staff – whether full time specialists, or part time staff, perhaps performing cross project SQA[3]. The audit itself is simple and straightforward because it is concerned simply with evaluating the extent of adherence to the policies. The policies, having been designed as concise statements that are either adhered to or not, are evaluated quickly. Arriving at such an answer will require some investigation, discussion and judgement (what constitutes 'reasonable' and 'up to date'), but this is quick and cheap compared to the usual audit process. The outcome of the audit is a list (or set of lists, if policies are categorized by process areas, or in some other way) of policy statements with corresponding yes/no checks. If desired some supporting commentary may be supplied

[2] These policies will also serve to justify any required process or product standards in use. See F - 1 Process Infrastructure.

[3] Cross project SQA is an attractive approach. Developers and managers are educated in the value of SQA and trained to audit. From time to time they audit work they are not involved in and others audit them. This is a low cost option that also ensures that managers and staff are aware of the value of an SQA function.

but is not essential. Issues arising can be dealt with in the normal way: dealt with at the lowest level and escalated as necessary.

4. Analysis: The policy audits are remarkably amenable to analysis. The responses can be counted and proportions of policies adhered to, presented for each category, and for each project, function, or department. Graphical presentation of these data is also straightforward. These analyses can be compared with performance figures, where these exist, or performance assessments, of development practices as required by policy, and conclusions drawn.

5. Action: The decisions to be taken based on this data, and the consequent value of the quality assurance function – this increased visibility – rests with the organization's senior management, as it does with a mature fully implemented SQA system.

With this simplified quality assurance system in place the value of policies, and a quality assurance system can be evaluated and revisions and adjustments made quickly and cheaply – either to policy to better meet organizational need or changing circumstances, or to the quality assurance system itself to make it function as required.

A natural development is the development of policy, elaborating on the initial statements, and the emergence of those standards, normally developed prematurely by a new SQA function, that help managers and development staff to satisfy policy requirements. If policy demands, for example, a reasonable plan, there will be requests for guidance on what constitutes a reasonable plan. A standard set of plan templates, maybe with supporting exemplars can be made available. A planning procedure may also be developed. These templates and procedures can be based on recognized best practice within the organization, or industry best practice or a mix of the two. However they originate they satisfy a real need, rather than being demanded in the name of unsubstantiated and unjustifiable best practice. Over time the quality assurance function may develop to require the use of these templates and procedures too, founded on demonstrated need and articulated by policy.

When policies are found to be overly restrictive, or made redundant by developing technology or tools they can be withdrawn.