CHAPTER 11

SOFTWARE QUALITY

Dolores Wallace* and Larry Reeker
National Institute of Standards and Technology
Gaithersburg, Maryland 20899 USA
{Dolores.Wallace, Larry.Reeker}@NIST.gov
*Dolores Wallace has retired from NIST (but is still available via her NIST e-mail address at the time of publication.)

Table of Contents

1. INTRODUCTION AND DEFINITION OF THE SOFTWARE QUALITY KNOWLEDGE AREA

This chapter deals with software quality considerations that transcend the life cycle processes. Of course, software quality is a ubiquitous concern in software engineering, so it is considered in many of the other KAs (and the reader will notice pointers those KAs through this KA. There will also be some inevitable duplication with those other KAs as a consequence.

Software Quality Assurance (SQA) and Verification and Validation (V&V) are the major processes discussed in this KA, as they bear directly on the quality of the software product. The term “product” will, however, be extended to mean any artifact that is the output of any process used to build the final software product. Examples of a product include, but are not limited to, an entire system specification, a software requirements specification for a software component of a system, a design module, code, test documentation, or reports from quality analysis tasks. While most treatments of quality are described in terms of the final system’s performance, sound engineering practice requires that intermediate products relevant to quality be checked throughout the development and maintenance process. The reason for this extension of “product” is that SQA and V&V can be used to evaluate the intermediate products and the final product. In addition to intermediate products and code, it can be applied to user documentation, which is best developed together with code and can often force issues regarding requirements and code.

Another major topic of this KA is just trying to answer the question “What is software quality?” this is not a simple question, as was concluded by David Garvin [Gar84, Hya96]. Though we will not go into the complexities that he studied, we will present a view for the working software engineer.

The discussion of the purpose and planning of SQA and V&V is a bridge between the discussion of quality and the activities and techniques discussion for SQA and V&V, but it is also an important activity in itself. In the planning process, the activities are designed to be fitted to the product and its purposes, including the quality attributes in the requirements.

Because determining quality of both the final product and intermediate products requires measurement, the topic of measurement is relevant to the other parts of this KA. A separate section is therefore included on the subject of measurement. Measurement of product quality at all levels of the project will in the future become more important than it has been in the past or is today. With increasing sophistication of systems (moving, for example, into areas like intelligent web agents), the questions of quality go beyond whether the system works or not, to how well it achieves measurable quality goals. In addition, the availability of more data about software and its production, along with data mining techniques for analysis of the data, will help to advance measurement definitions and procedures. A more relevant, widely-accepted, robust set of measures will be a sign of maturation in software engineering.

It has been suggested that this chapter should also deal with models and criteria that evaluate the capabilities of software organizations, but those are primarily project organization and management considerations. Of course it is not possible to disentangle the quality of the process from the quality of the product, but the quality of the software engineering process is not a topic specific to this KA, whereas the quality of the software product the assigned topic. So an ability to perform Software Quality Assurance, for instance, is a major component of a quality software engineering program, but SQA is itself relevant to software quality.

2. Breakdown of Topics for Software Quality

The quality of a given product is sometimes defined as "the totality of characteristics [of the product or services] that bear on its ability to satisfy stated or implied needs"[1]. Quality software is sometimes also defined as "the efficient, effective, and comfortable use by a given set of users for a set of purposes under specified conditions". These two definitions can be related to requirements conformance - provided the requirements are well engineered. Both agreement on quality requirements and communication to the engineer information on what will constitute quality requires that the aspects of quality be defined and discussed. For that reason, the first topic is description of product quality and some of the product characteristics that relate to it. The importance of requirements engineering is clearly an issue here.

Sections on the processes — SQA and V&V — that focus on software quality follow the discussion on software quality concepts. These quality-focused processes help to ensure better software in a given project. They also provide, as a by-product, general information to management that can improve the quality of the entire software and maintenance processes. The knowledge areas **Software Engineering Process** and **Software Engineering Management**, discuss quality programs for the organization developing software systems. SQA and V&V can provide relevant feedback for these areas.

Engineering for quality requires the measurement of quality in a concrete way, so this knowledge area contains a section on measurement as applied to SQA and V&V. Other processes for assuring software product quality are discussed in other parts of the SWEBOK. One of these, singled out as a separate KA within the software life cycle, **Software Testing**, is also used in both SQA and V&V.

2.1. Software quality concepts

What is software quality, and why is it so important that it is pervasive in the Software Engineering Body of Knowledge? Within a system, software is a tool, and tools have to be selected for quality and for appropriateness. That is the role of requirements. But software is more than a tool. The software dictates the performance of the system, and is therefore important to the system quality. Much thought must therefore go into the value to place on each quality attribute desired and on the overall quality of the system. This section discusses the value and the attributes of quality.

The notion of "quality" is not as simple as it may seem. For any engineered product, there are many desired qualities relevant to a particular project, to be discussed and determined at the time that the product requirements are determined. Quality attributes may be present or absent, or may be present in greater or lesser degree, with tradeoffs among them, with practicality and cost as major considerations. The software engineer needs first of all to determine the real purpose for the software, which is a prime point to keep in mind: The customer's needs come first, and they include particular levels of quality, not just functionality. Thus the software engineer has a responsibility to elicit quality requirements that may not even be explicit at the outset and to discuss their importance and the difficulty of attaining them. All processes associated with software quality (e.g. building, checking, improving quality) will be designed with these in mind and carry costs based on the design. Therefore, it is important to have in mind some of the possible attributes of quality.

- Various researchers have produced models (usually taxonomic) of software quality characteristics or attributes that can be useful for discussing, planning, and rating the quality of software products. The models often include measures to "measure" the degree of each quality attribute the product attains. They are not always direct measures of the quality characteristics discussed in the texts of Pressman [Pr], Pfleeger [Pf] and Kan [Kan94]. Each model may have a different set of attributes at the highest level of the taxonomy, and selection of and definitions for the - attributes at all levels may differ. The important point is that requirements define the required quality of the respective software, the definitions of the attributes for quality, and the measurement methods and acceptance criteria for the attributes. Some of the classical thinking in this area is found in McCall [McC77] and Boehm [Boe78].

2.1.1. Measuring the Value of Quality

A motivation behind a software project is a determination that it has a value, and this value may or not be quantified as a cost, but the customer will have some maximum cost in mind. Within that cost, the customer expects to attain the basic purpose of the software and may have some expectation of the necessary quality, or may not have thought through the quality issues or their related costs. The software engineer, in discussing software quality attributes and the processes necessary to assure them, should keep in mind the value of each attribute and the sensitivity of the value of the product to changes in it. Is it merely an adornment or is it essential to the system? If it is somewhere in between, as almost everything is, it is a matter of making the **customer** a part of the decision process and fully aware of both costs and benefits. Ideally, most of this decision process goes on in the Requirements phase (see that KA), but these issues may arise throughout the software life cycle. There is no definite rule for how the decisions are made, but the software engineer should be able to present quality alternatives and their costs. A

1 From *Quality—Vocabulary*, (ISO 8402: 1986, note 1).

discussion of measuring cost and value of quality requirements can be found in [Wei93], Chapter 8, pp118-134] and [Jon96], Chapter 5.

2.1.2. ISO 9126 Quality Description

Terminology for quality attributes differs from one taxonomy or model of software quality to another; each model may have different numbers of hierarchical levels and a different total number of attributes. A software engineer should understand the underlying meanings of quality characteristics regardless of their names, as well as their value to the system under development or maintenance. An attempt to standardize terminology in an inclusive model resulted in ISO 9126 (*Information Technology-Software Product Quality*, Part 1: Quality Model, 1998), of which a synopsis is included in this KA as Table 1. ISO 9126 is concerned primarily with the definition of quality characteristics in the final product. ISO 9126 sets out six quality characteristics, each very broad in nature. They are divided into 21 sub-characteristics. In the 1998 revision, "compliance" to application-specific requirements is included as a sub-characteristic of each characteristic The approach taken in the 1998 version is discussed in [Bev97].

2.1.3. Dependability

For systems whose failure may have extremely severe consequences, dependability of the overall system (hardware, software, and humans) is the main goal in addition to the realization of basic functionality. Software dependability is the subject of IEC 50-191 and the IEC 300 series of standards. Some types of systems (e.g., radar control, defense communications, medical devices) have particular needs for high dependability, including such attributes as fault tolerance, safety, security, usability. Reliability is a criterion under dependability and also is found among the ISO/IEC 9126 (Table 1). In Moore's treatment [M], Kiang's factors [Kia95] are used as shown in the following list, with the exception of the term Trustability from Laprie [Lap91].

- Availability: The product's readiness for use on demand
- Reliability: The longevity of product performance
- Maintainability: The ease of maintenance and upgrade
- Maintenance support: Continuing support to achieve availability performance objectives
- Trustability: System's ability to provide users with information about service correctness.

There is a large body of literature for systems that must be highly dependable ("high confidence" or "high integrity systems"). Terminology from traditional mechanical and electrical systems that may not include software have been imported for discussing threats or hazards, risks, system integrity, and related concepts, and may be found in the references cited for this section.

2.1.4. Special Types of Systems and Quality Needs

As implied above, there are many particular qualities of software that may or may not fit under ISO 9126. Particular classes of application systems may have other quality attributes to be judged. This is clearly an open-ended set, but the following are examples:

- Intelligent and Knowledge Based Systems – "Anytime" property (guarantees best answer that can be obtained within a given time if called upon for an answer in that amount of time), Explanation Capability (explains reasoning process in getting an answer).
- Human Interface and Interaction Systems – Adaptivity (to user's traits, interests), Intelligent Help, Display Salience.
- Information Systems – Ease of query, High recall (obtaining most relevant information), High Precision (not returning irrelevant information), tradeoffs. 3.5 Quality Attributes of Programming Products

Other considerations of software systems are known to affect the software engineering process while the system is being built and during its future evolution or modification, and these can be considered elements of product quality. These software qualities include, but are not limited to:

- "Stylishness" of Code
- Code and object reusability
- Traceability: From requirements to code/test documentation, and from code/test documentation to requirements
- Modularity of code and independence of modules.

These quality attributes can be viewed as satisfying organizational or project requirements for the software in the effort to improve the overall performance of the organization or project. See the Software Engineering Management and Software Engineering Process KAs for related material.

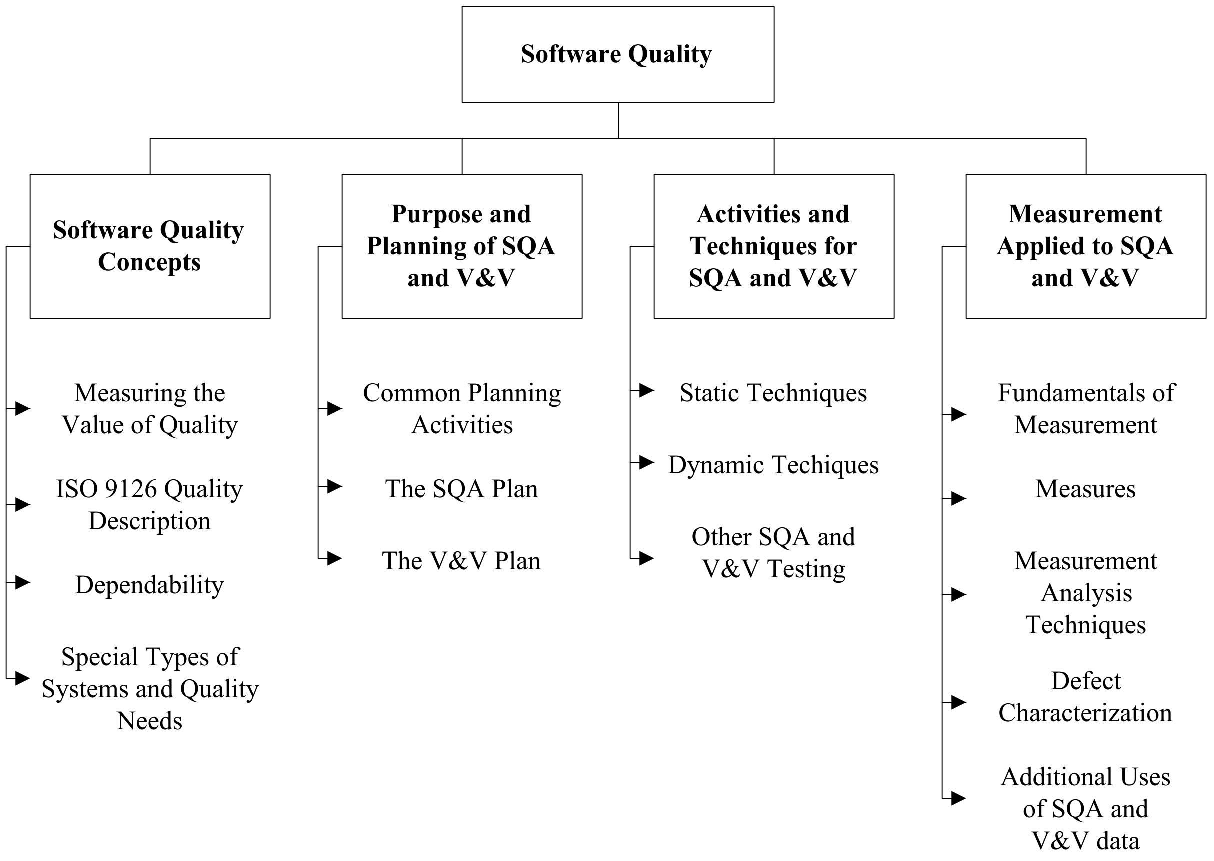
Software Quality
Software Quality Concepts
Measuring the Value of Quality
ISO 9126 Quality Description
Dependability
Special Types of Systems and Quality Needs
Purpose and Planning of SQA and V&V
Common Planning Activities
The SQA Plan
The V&V Plan
Activities and Techniques for SQA and V&V
Static Techniques
Dynamic Techiques
Other SQA and V&V Testing
Measurement Applied to SQA and V&V
Fundamentals of Measurement
Measures
Measurement Analysis Techniques
Defect Characterization
Additional Uses of SQA and V&V data

Table 1. Software Quality Characteristics and Attributes – ISO 9126-1998 View

Characteristics & Subcharacteristics	**Short Description of the Characteristics and Subcharacteristics**
Functionality	**Characteristics relating to achievement of the basic purpose for which the software is being engineered**
. Suitability	The presence and appropriateness of a set of functions for specified tasks
. Accuracy	The provision of right or agreed results or effects
. Interoperability	Software's ability to interact with specified systems
. Security	Ability to prevent unauthorized access, whether accidental or deliberate, to programs and data.
. Compliance	Adherence to application-related standards, conventions, regulations in laws and protocols
Reliability	**Characteristics relating to capability of software to maintain its level of performance under stated conditions for a stated period of time**
. Maturity	Attributes of software that bear on the frequency of failure by faults in the software
. Fault tolerance	Ability to maintain a specified level of performance in cases of software faults or unexpected inputs
. Recoverability	Capability and effort needed to reestablish level of performance and recover affected data after possible failure
. Compliance	Adherence to application-related standards, conventions, regulations in laws and protocols
Usability	**Characteristics relating to the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users**
. Understandability	The effort required for a user to recognize the logical concept and its applicability
. Learnability	The effort required for a user to learn its application, operation, input, and output
. Operability	The ease of operation and control by users
. Attractiveness	The capability of the software to be attractive to the user
. Compliance	Adherence to application-related standards, conventions, regulations in laws and protocols
Efficiency	**Characteristic related to the relationship between the level of performance of the software and the amount of resources used, under stated conditions**
. Time behavior	The speed of response and processing times and throughput rates in performing its function
. Resource utilization	The amount of resources used and the duration of such use in performing its function
. Compliance	Adherence to application-related standards, conventions, regulations in laws and protocols
Maintainability	**Characteristics related effort needed to make modifications, including corrections, improvements or adaptation of software to changes in environment, requirements and functional specifications**

Table 1. Software Quality Characteristics and Attributes – ISO 9126-1998 View

Characteristics & Subcharacteristics	**Short Description of the Characteristics and Subcharacteristics**
. Analyzability	The effort needed for diagnosis of deficiencies or causes of failures, or for identification parts to be modified
. Changeability	The effort needed for modification fault removal or for environmental change
. Stability	The risk of unexpected effect of modifications
. Testability	The effort needed for validating the modified software
. Compliance	Adherence to application-related standards, conventions, regulations in laws and protocols
Portability	**Characteristics related to the ability to transfer the software from one organization or hardware or software environment to another**
. Adaptability	The opportunity for its adaptation to different specified environments
. Installability	The effort needed to install the software in a specified environment
. Co-existence	The capability of a software product to co-exist with other independent software in common environment
. Replaceability	The opportunity and effort of using it in the place of other software in a particular environment
. Compliance	Adherence to application-related standards, conventions, regulations in laws and protocols

2.2. Purpose and Planning of SQA and V&V

The KA **Software Requirements** describes how the requirements and their individual features are defined, prioritized and documented and how the quality of that documentation can be measured. The set of requirements has a direct effect on both the intermediate software engineering products, and the delivered software. Building in quality as the process takes place and making careful reference to well-engineered requirements that define the needed measures and attributes of quality are the most important determiners of overall software quality.

The **Software Engineering Process** (discussed overall in that KA) employs multiple *supporting processes* to examine and assure software products for quality. These supporting processes conduct activities to ensure that the software engineering process required by the project is followed. Two related (and sometimes combined) supporting processes most closely related to product quality, SQA and V&V, are discussed in this section. These processes both encourage quality and find possible problems. But they differ somewhat in their emphasis.

SQA and V&V also provide management with visibility into the quality of products at each stage in their development or maintenance. The visibility comes from the data and measurements produced through the performance of tasks to assess and measure quality of the outputs of any software life cycle processes as they are developed. Where strict quality standards are an overriding factor, the tasks used to assess quality and capture data and measurements may be performed by an organization independent of the project organization, in order to provide a higher degree of objectivity to the quality assessment.

The SQA process provides assurance that the software products and processes in the project life cycle conform to their specified requirements by planning a set of activities to help build quality into the software. This means ensuring that the problem is clearly and adequately stated and that the solution's requirements are properly defined and expressed. SQA seeks to retain the quality throughout the development and maintenance of the product by execution of a variety of activities at each stage that can result in early identification of problems, which are almost inevitable in any complex activity. The SQA role with respect to process is to ensure that planned processes are appropriate and are later implemented according to plan and that relevant measurement processes are provided to the appropriate organization.

The Verification and Validation process determines whether products of a given development or maintenance activity conform to the needs of that activity and those imposed by previous activities, and whether the final software product satisfies its intended use and user needs. Verification attempts to ensure that the product is built correctly, in the sense that the, output products of an activity fulfill requirements imposed on them in previous activities. Validation attempts to ensure that the right product is built, that is, the product fulfills its specific intended use. Both verification and validation processes begin early in the development or maintenance process. They provide an examination of every product relative both to its immediate predecessor and to the system requirements it must satisfy.

In summary, the SWEBOK describes a number of pro ways of achieving software quality. As described in this KA, the SQA and V&V processes are closely related processes that can overlap and are sometimes even combined. They seem largely reactive in nature because they address the processes as practiced and the products as produced; but they have a major role at the planning stage in being proactive as to the procedures needed to attain the quality attributes and degree needed by the stakeholders in the software. They should also produce feedback that can improve the software engineering process. In summary:

- SQA governs the procedures meant to build the desired quality into the products by assuring that the process is well-planned and then applied as prescribed and defined. It helps keep the organization from sliding back into less effective processes and habits, and may provide direct assistance or guidance in applying the current practices.

- V&V is aimed more directly at product quality, in that it is based on testing that can locate deviations and fix them. But it also validates the intermediate products and therefore the intermediate steps of the software engineering process. So it too can affect the software engineering process through that evaluation.

It should be noted that sometimes the terms SQA and V&V are associated with organizations rather than processes. SQA often is the name of a unit within an organization. Sometimes an independent organization is contracted to conduct V&V. Testing may occur in Both SQA and V&V and is discussed in this KA in relation to those processes. Details on testing within the software life cycle are found in the KA on **Software Testing**. The Software Quality KA is not intended to define organizations but rather the purposes and procedures of SQA and V&V, insofar as they relate to software quality. The organizational aspect is mentioned here, however, to tie together different KAs and to help avoid confusion. Some discussion on organizational issues appears in [Hum98], and the IEEE Std. 1012.

2.2.1. Common Planning Activities

Planning for software quality involves (1) defining, the required product in terms of its quality attributes and (2) planning the processes to achieve the required product. Planning of these processes is discussed in other KAs: **Software Engineering Management**, **Software Engineering Design**, and **Software Engineering Methods and Tools**. These topics are different from planning the SQA and V&V processes. The SQA and V&V processes assess predicted adequacy and actual implementation of those plans, that is, how well software products will or do satisfy customer and stakeholder requirements, provide value to the customers and other stakeholders, and meet the software_quality needed to meet the system requirements.

System requirements vary among systems, as do the activities selected from the disciplines of SQA and V&V. Various factors influence planning, management and selection of activities and techniques, including:

1. the environment of the system in which the software will reside;
2. system and software requirements;
3. the commercial or standard components to be used in the system;
4. the specific software standards used in developing the software;
5. the software standards used for quality;
6. the methods and software tools to be used for development and maintenance and for quality evaluation and improvement;
7. the budget, staff, project organization, plans and schedule (size is inherently included) of all the processes;
8. the intended users and use of the system, and
9. the integrity level of the system.

Information from these factors influences how the SQA and V&V processes are organized, and documented, how specific SQA and V&V activities are selected, and what resources are needed or will impose bounds on the efforts. The integrity level of a system can be used as an example. The integrity level is determined based on the possible consequences of failure of the system and the probability of failure. For software systems where safety or security is important, techniques such as hazard analysis for safety or threat analysis for security may be used to develop a planning process that would identify where potential trouble spots lie. Failure history of similar systems may also help in identifying which activities will be most useful in detecting faults and assessing quality.

If the SQA and V&V organizations are the same, their plans may be combined, but we will treat them as separate plans below, as they are often distinguished from one another.

2.2.2. The SQA Plan

The SQA plan defines the processes and procedures that will be used to ensure that software developed for a specific product meets its requirements and is of the highest quality possible within project constraints. To do so, it must first ensure that the quality target is clearly defined and understood. The plan may be governed by software quality assurance standards, life cycle standards, quality management standards and models, company policies and procedures for quality and quality improvement. It must consider management, development and maintenance plans for the software. Standards and models such as ISO9000, CMM, Baldrige, SPICE, TickIT are related to the **Software Engineering Process** and may influence the SQA plan.

The specific activities and tasks are laid our, with their costs and resource requirements, their overall management, and their schedule in relation to those in the software management, development or maintenance plans. The SQA plan should be cognizant of the software configuration plan also (see the KA for **Software Configuration Management**) The SQA plan identifies documents, standards, practices, and conventions that govern the project and how they will be checked and monitored to ensure adequacy or compliance. The SQA plan identifies measures, statistical techniques, procedures for problem reporting and corrective action, resources such as tools, techniques and methodologies, security for physical media, training, and SQA reporting and documentation to be retained. The SQA plan addresses assurance of any other type of function addressed in the software plans, such as supplier software to the project or commercial off-the-shelf software (COTS), installation, and service after delivery of the system. It can also contain some items less directly related to quality: acceptance criteria, activity deadlines, reporting, and management activities that feed experiences into the development process.

2.2.3. The V&V Plan

The V&V plan is the instrument to explain the requirements and management of V&V and the role of each technique in satisfying the objectives of V&V. An understanding of the different purposes of each verification and validation activity will help in planning carefully the techniques and resources needed to achieve their purposes. IEEE standard 1012, section 7, specifies what ordinarily goes into a V&V plan.

Verification activities examine a specific product, that is, output of a process, and provide objective evidence that specified requirements have been fulfilled. The "specified requirements" refer to the requirements of the examined product, relative to the product from which it is derived. For example, code is examined relative to requirements of a design description, or the software requirements are examined relative to system requirements.

Validation examines a specific product to provide objective evidence that the requirements for a specific intended use are fulfilled. The validation confirms that the product traces back to the software system requirements and satisfies them. This includes planning for system testing more or less in parallel with the system and software requirements process. This aspect of validation often serves as part of a requirements verification activity. While some communities separate completely verification from validation, the activities of each actually service the other.

V&V activities can be exercised at every step of the life cycle, often on the same product, possibly using the same techniques in some instances. The difference is in the technique's objectives for that product, and the supporting inputs to that technique. Sequentially, verification and validation will provide evidence from requirements to the final system, a step at a time. This process holds true for any life cycle model, gradually iterating or incrementing through the development. The process holds in maintenance also.

The plan for V&V addresses the management, communication, policies and procedures of the V&V activities and their iteration, evaluation of methods, measures, and tools for the V&V activities, defect reports, and documentation requirements. The plan describes V&V activities, techniques and tools used to achieve the goals of those activities.

The V&V process may be conducted in various organizational arrangements. First, to re-emphasize, many V&V techniques may be employed by the software engineers who are building the product. Second, the V&V process may be conducted in varying degrees of independence from the development organization. Finally, the integrity level of the product may drive the degree of independence.

2.3. Activities and techniques for SQA and V&V

The SQA and V&V processes consist of activities to indicate how software plans (e.g., management, development, configuration management) are being implemented and how well the evolving and final products are meeting their specified requirements. Results from these activities are collected into reports for management before corrective actions are taken. The management of SQA and V&V are tasked with ensuring the quality of these reports, that is, that the results are accurate.

Specific techniques to support the activities software engineers perform to assure quality may depend upon their personal role (e.g., programmer, quality assurance staff) and project organization (e.g., test group, independent V&V). To build or analyze for quality, the software engineer understands development standards and methods and the genesis of other resources on the project (e.g., components, automated tool support) and how they will be used. The software engineer performing quality analysis activities is aware of and understands considerations affecting quality assurance: standards for software quality assurance, V&V, testing, the various resources that influence the product, techniques, and measurement (e.g., what to measure and how to evaluate the product from the measurements).

The SQA and V&V activities consist of many techniques; some may directly find defects and others may indicate where further examination may be valuable. These may be referred to as direct-defect finding and supporting techniques. Some often serve as both, such as people-intensive techniques like reviews, audits, and inspection (as used here, not to be confused with the term "inspection" used for static analysis of work products) and some static techniques like complexity analysis and control flow analysis. The SQA and V&V techniques can be categorized as two types: static and dynamic. Static techniques do not involve the execution of code, whereas dynamic techniques do. Static techniques involve examination of the documentation (e.g., requirements specification, design, plans, code, test documentation) by individuals or groups of individuals and sometimes with the aid of automated tools. Often, people tend to think of testing as the only dynamic technique, but simulation is an example of another one. Sometimes static techniques are used to support dynamic techniques, and vice-versa. An individual, perhaps with the use of a software tool, may perform some techniques; in others, several people are required to conduct the technique. Such techniques, requiring two or more people, are "people-intensive". Depending on project size, other techniques, such as testing, may involve many people, but are not people-intensive in the sense described here.

Static and dynamic techniques are used in either SQA or V&V. Their selection, specific objectives and organization depend on project and product requirements. Discussion in the following sections and the tables in the appendices provide only highlights about the various techniques; they are not inclusive. There are too many techniques to define in this document but the lists and references provide a flavor of SQA and V&V techniques and will yield insights for selecting techniques and for pursuing additional reading about techniques.

2.3.1. Static Techniques

Static techniques involve examination of the project's documentation, software and other information about the software products without executing them. The techniques may include people intensive activities, as defined above, or analytic activities conducted by individuals, with or without the assistance of automated tools. These support both SQA and V&V processes and their specific implementation can serve the purpose of SQA, verification, or validation, at every stage of development or maintenance.

2.3.1.1. People-Intensive Techniques

The setting for people-intensive techniques, including audits, reviews, and inspections, may vary. The setting may be a formal meeting, an informal gathering, or a desk-check situation, but (usually, at least) two or more people are involved. Preparation ahead of time may be necessary. Resources in addition to the items under examination may include checklists and results from analytic techniques and testing. Another technique that may be included in this group is the walkthrough. They may also be done on-line. These activities are discussed in IEEE Std. 1028 on reviews and audits, [Fre82], [Hor96], and [Jon96], [Rak97].

Reviews that specifically fall under the SQA process are technical reviews, that is, on technical products. However, the SQA organization may be asked to conduct management reviews as well. Persons involved in the reviews are usually a leader, a recorder, technical staff, and -in the management review - management staff.

Management reviews determine adequacy of and monitor progress or inconsistencies against plans and schedules and requirements. These reviews may be exercised on products such as audit reports, progress reports, V&V reports and plans of many types including risk management, project management, software configuration management, software safety, and risk assessment, among others. See the **Software Engineering Management** KA for related material.

Technical reviews examine products (again, anything produced a stage of the software engineering project, such as software requirement specifications, software design documents, test documentation, user documentation, installation procedures), but the coverage of the material may vary with purpose of the review. The subject of the review is not necessarily the completed product, but may be a portion of it. For example, a subset of the software requirements may be reviewed for a particular set of functionality, or several design modules may be reviewed, or separate reviews may be conducted for each category of test for each of its associated documents (plans, designs, cases and procedures, reports).

An audit is an independent evaluation of conformance of software products and processes to applicable regulations, standards, plans, and procedures. Audits may examine plans like recovery, SQA, and maintenance, design documentation. The audit is a formally organized activity, with participants having specific roles, such as lead auditor, other auditors, a recorder, an initiator, and a representative of the audited organization. While for reviews and audits there may be many formal names such as those identified in the IEEE Std. 1028, the important point is that they can occur on almost any product at any stage of the development or maintenance process.

Software inspections generally involve the author of a product, while reviews likely do not. Other persons include a reader and some inspectors. The inspector team may consist of different expertise, such as domain expertise, or design method expertise, or language expertise, etc. Inspections are usually conducted on a relatively small section of the product. Often the inspection team may have had a few hours to prepare, perhaps by applying an analytic technique to a small section of the product, or to the entire product with a focus only on one aspect, e.g., interfaces. A checklist, with questions germane to the issues of interest, is a common tool used in inspections. Inspection sessions can last a couple of hours or less, whereas reviews and audits are usually broader in scope and take longer.

The walkthrough is similar to an inspection, but is conducted by only members of the development group, who examine a specific part of a product. With the exception of the walkthrough – primarily an assurance technique used only by the developer, these people-intensive techniques are traditionally considered to be SQA techniques, but may be performed by others. The technical objectives may also change, depending on who performs them and whether they are conducted as verification or as validation activities. Often, when V&V is an organization, it may be asked to support these techniques, either by previous examination of the products or by attending the sessions to conduct the activities.

2.3.1.2 Analytic Techniques

An individual generally applies analytic techniques. Sometimes several people may be assigned the technique, but each applies it to different parts of the product. Some are tool-driven; others are primarily manual. With the References (Section 7.1) there are tables of techniques according to their primary purpose. However, many techniques listed as support may find some defects directly but are typically used as support to other techniques. Some however are listed in both categories because they are used either way. The support group of techniques also includes various assessments as part of overall quality analysis. Examples of this group of techniques includes complexity analysis, control flow analysis, algorithm analysis, and use of formal methods.

Each type of analysis has a specific purpose and not all are going to be applied to every project. An example of a support technique is complexity analysis, useful for determining that the design or code may be too complex to develop correctly, to test or maintain; the results of a complexity analysis may be used in developing test cases. Some listed under direct defect finding, such as control flow analysis, may also be used as support to another activity. For a software system with many algorithms, algorithm analysis is important, especially when an incorrect algorithm could cause a catastrophic result. There are too many analytic techniques to define in this document but the lists and references provide a flavor of software analysis and will yield to the software engineer insights for selecting techniques and for pursuing additional reading about techniques.

A class of analytic techniques that is gaining greater acceptance is the use of formal methods to verify software requirements and designs. Proof of correctness may also be applied to different parts of programs. Their acceptance to date has mostly been in verification of crucial parts of critical systems, such as specific security and safety requirements [NAS97].

2.3.2. Dynamic Techniques

Different kinds of dynamic techniques are performed throughout the development and maintenance of software systems. Generally these are testing techniques, but techniques such as simulation, model checking, and symbolic execution may be considered dynamic. Code reading is considered a static technique but experienced software engineers may execute the code as they read through it. In this sense, code reading may also fit under dynamic. This discrepancy in categorizing indicates that people with different roles in the organization may consider and apply these techniques differently.

Some testing may fall under the development process, the SQA process, or V&V, again depending on project organization. The discipline of V&V encompasses testing and requires activities for testing at the very beginning of the project. Because both the SQA and V&V plans address testing, this section includes some commentary about testing. The knowledge area on **Software Testing** provides discussion and technical references to theory, techniques for testing, and automation. Supporting techniques for testing fall under test management, planning and documentation. V&V testing generally includes component or module, integration, system, and acceptance testing. V&V testing may include test of commercial off-the-shelf software (COTS) and evaluation of tools to be used in the project (see section 5.3).

The assurance processes of SQA and V&V examine every output relative to the software requirement specification to ensure the output's traceability, consistency, completeness, correctness, and performance. This confirmation also includes exercising the outputs of the development and maintenance processes, that is, the analysis consists of validating the code by testing to many objectives and strategies, and collecting, analyzing and measuring the results. SQA ensures that appropriate types of tests are planned, developed, and implemented, and V&V develops test plans, strategies, cases and procedures.

2.4. Other SQA and V&V Testing

Two types of testing fall under SQA and V&V because of their responsibility for quality of materials used in the project:

Evaluation and test of tools to be used on the project (See ISO/IEC 12119 Information Technology – Guidance for the Evaluation and Selection of CASE Tools)

Conformance test (or review of conformance test) of components and COTS products to be used in the product. There now exists a standard for software packages (see section 7.2.4.)

The SWEBOK knowledge area on **Software Testing** addresses special purpose testing. Many of these types are also considered and performed during planning for SQA or V&V testing. Occasionally the V&V process may be asked to perform these other testing activities according to the project's organization. Sometimes an independent V&V organization may be asked to monitor the test process and sometimes to witness the actual execution, to ensure that it is conducted in accordance with specified procedures. And, sometimes, V&V may be called on to evaluate the testing itself: adequacy of plans and procedures, and adequacy and accuracy of results.

Another type of testing that may fall under a V&V organization is third party testing. The third party is not the developer or in any way associated with the development of the product. Instead, the third party is an independent facility, usually accredited by some body of authority. Their purpose is to test a product for conformance to a specific set of requirements. Discussion on third party testing appears in the July/August 1999 *IEEE Software* special issue on software certification.

2.5. Measurement applied to SQA and V&V

SQA and V&V discover information at all stages of the development and maintenance process that provides visibility into the software development and maintenance processes. Some of this information involves counting and classifying defects, where "defect" refers to errors, faults, and failures. Typically, if the word "defect" is used, it refers to "fault" as defined below, but different cultures and standards may differ somewhat in their meaning for these same terms, so there have been attempts to define them. Partial definitions taken from the IEEE Std 610.12-1990

("IEEE Standard Glossary of Software Engineering Terminology") are these:

- Error: "A difference…between a computed result and the correct result"
- Fault: "An incorrect step, process, or data definition in a computer program"
- Failure: "The [incorrect] result of a fault"
- Mistake: "A human action that produces an incorrect result".

Mistakes (as defined above) are the subject of the quality improvement process, which is covered in the Knowledge Area **Software Engineering Process**. Failures found in testing as the result of software faults are included as defects in the discussion of this section. Reliability models are built from failure data collected during system testing or from systems in service, and thus can be used to predict failure and to assist decisions on when to stop testing.

Information on inadequacies and defects found during SQA and V&V techniques may be lost unless it is recorded. For some techniques (e.g., reviews, audits, inspections), recorders are usually present to record such information, along with issues, and decisions. When automated tools are used, the tool output may provide the defect information. Sometimes data about defects are collected and recorded on a "trouble report" form and may further be entered into some type of database, either manually or automatically from an analysis tool. Reports about the defects are provided to the software management and development organizations.

One probable action resulting from SQA and V&V reports is to remove the defects from the product under examination. Other actions enable achieving full value from the findings of the SQA and V&V activities. These actions include analyzing and summarizing the findings with use of measurement techniques to improve the product and the process ands to track the defects and their removal. Process improvement is primarily discussed in **Software Engineering Process** with SQA and V&V process being a source of information..

2.5.1. Fundamentals of Measurement

The theory of measurement establishes the foundation on which meaningful measurements can be made. It tells us, for instance, that the statement that it is twice as warm today as yesterday if it is 40 degrees Fahrenheit today but only 20 degrees yesterday is not meaningful because degrees Fahrenheit is not a "ratio scale" but a similar statement concerning degrees Kelvin would have a physical meaning. Measurement is defined in the theory as "the assignment of numbers to objects in a systematic way to represent properties of the object." If the property is just a constant assigned by counting some aspect it is an "absolute" measure, but usually not very meaningful. More meaningful scales are relative to a classification or scale, and for those, measurement theory provides a succession of more and more constrained ways of assigning the measures. If the numbers assigned are merely to provide labels to classify the objects, they are called "nominal". If they are assigned in a way that ranks the objects (e.g. good, better, best), they are called "ordinal". If they deal with magnitudes of the property relative to a defined measurement unit, they are "interval" (and the intervals are uniform between the numbers unless otherwise specified, and are therefore additive). Measurements are at the "ratio" level if they have an absolute zero point, so ratios of distances to the zero point are meaningful (as in the example of temperatures given earlier).

Key terms on software measures and measurement methods have been defined in ISO/IEC FCD 15939 on the basis of the ISO international vocabulary of metrology [ISO93]. Nevertheless, readers will encounter terminology differences in the literature; for example, the term "metric" is sometimes used in place of "measure".

Software measures of all of these types have been defined. A simple example of a ratio scale in software, for instance, is the number of defects discovered per module. In module 1, there may be 10 defects per function point (where a function point is a measure of size based on functionality) in module 2, 15 and in module 3, 20. The difference between module 1 and 2 is 5 and module 3 has twice as many defects as module 1. Theories of measurement and scales are discussed in [Kan94], pp. 54-82. The standard for functional size measurement is ISO/IEC 14143-1 and additional, supporting standards are under development. A number of specific methods, suitable for different purposes, are available.

Measurement for measurement's sake does not help define quality. Instead, the software engineer needs to define specific questions about the product, and hence the objectives to be met to answer those questions. Only then can specific measures be selected. ISO/IEC FCD 15939 defines the activities and tasks necessary to implement a software measurement process and includes as well a measurement information model. Another approach is "Plan-Do-Check-Act" discussed in [Rak97] . Others are discussed in the references on software measurement. The point is that there has to be a reason for collecting data, that is, there is a question to be answered.

Measurement programs are considered useful if they help project stakeholders (1) understand what is happening during their processes, and (2) control what is happening on their projects [Fen95,97, Pf]. For measurement to work well, it is critical to establish measurement planning, collection, interpretation and reporting activities as part of a larger organizational process, for example requirements engineering, design, or software construction. The measurement process and its implementation should be documented in the form of a measurement plan. It defines the measurement process with exact information on stakeholders involved, measurement frequency, sources of measurement data, measurement rules, measurement data interpretation rules, tools support, reports to be produced, and action items that can be taken based on the measurement data. In this way, the plan represents a communication vehicle to ensure that all team members agree with the measurement approach, while also serving as the ongoing reference model to manage the implementation of reuse measures.

Other important measurement practices deal with experimentation and data collection. Experimentation is useful in determining the value of a development, maintenance, or assurance technique and results may be used to predict where faults may occur. Data collection is non-trivial and often too many types of data are collected. Instead, it is important to decide what is the purpose, that is, what question is to be answered from the data, then decide what data is needed to answer the question and then to collect only that data. While a measurement program has costs in time and money, it may result in savings. Methods exist to help estimate the costs of a measurement program. Discussion on the following key topics for measurement planning are found in ([Bas84], [Kan94], [Pr], [Pf], [Rak97], [Zel98]:

- Experimentation
- Selection of approach for measurement
- Methods
- Costing
- Data Collection process.

2.5.2. Measures

Measurement models and frameworks for software quality enable the software engineer to establish specific product measures as part of the product concept. Models and frameworks for software quality are discussed in [Kan94], [Pf], and [Pr].

If they are designed properly measures can support software quality (among other aspects of the software engineering process) in multiple ways. They can help management decision-making. They can find problematic areas and bottlenecks in the software product; and they can help the developers in assessing the quality of their work for SQA purposes and for longer term process quality assessment.

Data can be collected on various characteristics of software products. Many of the measures are related to the quality characteristics defined in **Section 2** of this Knowledge Area. Much of the data can be collected as results of the **static techniques** previously discussed and from various testing activities (see **Software Testing** Knowledge Area). The types of measures for which data are collected generally fall into one or more of these categories and are discussed in [Jon96], [Lyu96], [Pf], [Pr], [Lyu96], and [Wei93]:

- Quality characteristics measures
- Reliability models & measures
- Defect features (e.g., counts, density)
- Customer satisfaction
- Product features (e.g., size, which includes source lines of code)and/or function points [Abr96], number of requirements)
- Structure measures (e.g., modularity, complexity, control flow)
- Object-oriented measures.

2.5.3. Measurement Analysis Techniques

While the measures for quality characteristics and product features may be useful in themselves (for example, the number of defective requirements or the proportion of requirements that are defective), mathematical and graphical techniques can be applied to aid in interpretation of the measures. These fit into the following categories and are discussed in [Fen97], [Jon96], [Kan94], [Lyu96] and [Mus98].

- Statistically based (e.g., Pareto analysis, run charts, scatter plots, normal distribution)
- Statistical tests (e.g., binomial test; chi-squared test)
- Trend analysis
- Prediction, e.g., reliability models.

The statistically based techniques and tests often provide a snapshot of the more troublesome areas of the software product under examination. The resulting charts and graphs are visualization aids that the decision-makers can use to focus resources where they appear most needed. Results from trend analysis may indicate whether a schedule may be slipped, such as in testing, or may indicate that certain classes of faults will gain in intensity unless some corrective action is taken in development. And the predictive techniques assist in planning test time and predicting failure. More discussion on these appears in **Software Engineering Process** and **Software Engineering Management**.

2.5.4. Defect Characterization

SQA and V&V processes discover defects. Characterizing those defects enables understanding of the product, facilitates corrections to the process or the product, and informs the project management or customer of the status of the process or product. Many defect (fault) taxonomies exist and while attempts have been made to get consensus on a fault and failure taxonomy, the literature indicates that quite a few are in use (IEEE Std. 1044, [Bei90], [Chi92], [Gra92]). Defect (anomaly) characterization is used in audits and reviews, too, with the review leader often presenting a list of anomalies provided by team members for consideration at a review meeting.

As new design methodologies and languages evolve, along with advances in overall application technologies, new classes of defects appear, or, the connection to previously defined classes requires much effort to realize. When tracking defects, the software engineer is interested not only in the count of defects, but the types. Without some classification, information will not really be useful in identifying the underlying causes of the defects because no one will be able to group specific types of problems and make determinations about them. The point, again, as in selecting a measurement approach with quality characteristics, measures and measurement techniques, is to establish a defect taxonomy that is meaningful to the organization and software system.

The above references as well as [Kan94], [Fen95] and [Pf], and [Jon89] all provide discussions on analyzing defects. This is done by measuring defect occurrences and then applying statistical methods to understand the types of defects that occur most frequently, that is, answering questions about where mistakes occur most frequently (their density). They also aid in understanding the trends and how well detection techniques are working, and, how well the development and maintenance processes are doing.[2] Measuring test coverage helps to estimate how much test effort remains and to predict possible remaining defects. From these measurement methods, one can develop defect profiles for a specific application domain. Then, for the next software system within that organization, the profiles can be used to guide the SQA and V&V processes, that is, to expend the effort where the problems are likeliest to occur. Similarly, benchmarks, or defect counts typical of that domain, may serve as one aid in determining when the product is ready for delivery.

The following topics are useful for establishing measurement approaches for the software products:

- Defect classification and descriptions
- Defect analysis
- Measuring adequacy of the SQA and V&V activities
- Test coverage
- Benchmarks, profiles, baselines, defect densities.

2.5.5. Additional Uses of SQA and V&V data

The measurement section of this KA on SQA and V&V touches only minimally on measurement, for measurement is a major topic itself. The purpose here is only to provide some insight on how the SQA and V&V processes use measurement directly to support achieving their goals. There are a few more topics which measurement of results from SQA and V&V may support. These include some assistance in deciding when to stop testing. Reliability models and benchmarks, both using fault and failure data, are useful for this objective. Again, finding a defect, or perhaps trends among the defects, may help to locate the source of the problem.

[2] Discussion on using data from SQA and V&V to improve development and maintenance processes appears in **Software Engineering Management** and **Software Engineering Process**.

The cost of SQA and V&V processes is almost always an issue raised in deciding how to organize a project. Often generic models of cost, based on when the defect is found and how much effort it takes to fix the defect relative to finding the defect earlier, are used. Data within an organization from that organization's projects may give a better picture of cost for that organization. Discussion on this topic may be found in [Rak97], pp. 39-50. Related information can be found in the **Software Engineering Process** and_**Software Engineering Management** KAs.

Finally, the SQA and V&V reports themselves provide valuable information not only to these processes but to all the other software engineering processes for use in determining how to improve them. Discussions on these topics are found in [McC93] and IEEE Std. 1012.

3. BREAKDOWN RATIONALE

One breakdown of topics is provided for this area. The rationale for that breakdown is largely stated in the KA introduction. This has been developed through an evolutionary process as the various rewrites and review cycles took place.

The original name of the topic, as it came out of the first meeting of the Industrial Review Board, was "Software Quality Analysis, and it had resulted from a fusion of

- Software Quality Assurance
- Verification and Validation
- Dependability and Quality
- The jump-start document (produced by the same authors as this current KA version) suggested three breakdowns . They were based on
- Criteria for Quality of Software (Basic General Criteria, Examples of Implicit Requirements, Special Situations with Additional Quality Criteria)
- Maintaining and Improving Quality in Software (Process or Project Quality, Product Quality, Techniques for Effective V&V)
- Verification and Validation Across the Software Life Cycle (Initial Project V&V Management, Software Requirements V&V, Software Design V&V, Coding V&V, Testing Phase)

It soon became clear that the topic was intended to transcend life cycle divisions, and that the third suggested breakdown could be covered by references to the KAs covering stages of the life cycle. The first two breakdowns did not really have major overlaps, but each dealt with topics that related to quality, so they were merged into a single breakdown.

An attempt to define the title "Software Quality Analysis" was included in early versions, and it distinguished Quality Process and Quality Product. The Product portion dwelt in some detail on views of quality characteristics. The Process section included SQA and V&V and some management-oriented considerations.

Later it was determined that the management portions were covered well elsewhere in the SWEBOK, and that the purpose of this KA was really Quality Product. Other KAs were describing the process, including quality concerns, in their descriptions. Nevertheless, there was a place for the processes (SQA and V&V) whose major concern was quality, as this would pull together fragmented discussions in the life cycle KAs and emphasize that these processes were in principle the same over all stages.

Since the ISO 9126 characteristics are well set out in the standard, and there are other views of quality characteristics as well, the detailed examination of them that appeared in earlier versions has also been reduced and dealt with through references. This was suggested by reviewers and by space considerations.

In summary, the breakdown is a product of the original concept of the editorial team; the suggestions of the Industrial Advisory Board; the material developed by other KA authors; and the opinions voiced by dozens of individuals, representing different points of view, who have reviewed this KA. During the process, the word "Analysis" was dropped from the KA title, since it was causing confusion as to the purpose of the KA by implying to some readers a scholarly area, rather than an area of concern to the practitioner.

It is intended that the KA as a whole and its breakdown of the topic will now evolve based on experience by users, reflecting its usefulness in fulfilling the multiple objectives of the SWEBOK.

4. Matrix of Topics vs. Reference material

Software Quality Concepts	[Boe78]	[D]	[Fen97]	[Kia95]	[Lap91]	[Lew92]	[Lyu96]	[M]	[Mus98]	[Pf]	[Pr]	[Rak97]	[S]	[Wal96]	[Wei93]
Value of Quality	X									X					X
Functionality												X			
Reliability							X	X	X	X	X	X	X	X	
Efficiency							X	X			X				
Usability			X			X		X		X	X	X	X		
Maintainability			X	X		X		X		X	X		X		
Portability								X		X	X	X	X		
Dependability			X		X		X	X	X	X	X		X	X	
Other Qualities		X				X		X		X	X		X	X	

Definition & Planning for Quality	[Gra92]	[Hor96]	[Kaz99]	[Lew92]	[Lyu96]	[McC93]	[M]	[Mus98]	[Pf]	[Pr]	[Rak97]	[Sch98]	[S]	[Wal89]	[Wal96]
Overall							X		X	X			X		
SQA		X		X		X	X		X	X	X	X	X		
VV				X			X		X	X	X	X	X	X	X
Independent V&V				X					X	X	X		X	X	X
Hazard, threat anal.							X		X	X			X		X
Risk assessment	X	X		X	X		X	X					X		
Performance analysis			X						X	X					

Techniques Requiring Two or More People	[Ack97]	[Ebe94]	[Fre82]	[Gra92]	[Hor96]	[Lew92]	[McC93]	[Pf]	[Pr]	[Rak97]	[Sch98]	[S]	[Wal89]	[Wal96]
Audit			X		X	X				X			X	
Inspection	X	X	X	X	X		X	X	X	X	X	X	X	X
Review			X		X	X	X	X	X			X	X	X
Walkthrough			X		X		X	X	X			X	X	X

Support to Other Techniques	[Bei90]	[Con86]	[Fri95]	[Het84]	[Lev95]	[Lew92]	[Lyu96]	[Mus98]	[Pf]	[Pr]	[Rak97]	[Rub94]	[S]	[Fri95]	[Wal89]	[Wal96]
Change Impact Anal.							X		X	X			X			
Checklists				X		X					X	X				
Complexity Analysis	X	X					X	X		X						
Coverage Analysis	X						X	X								
Consistency Analysis									X	X		X				
Criticality Analysis					X	X				X						X
Hazard Analysis			X		X					X	X		X			
Sensitivity Analysis			X											X		
Slicing	X													X		X
Test documents	X	X					X	X							X	X
Tool evaluation						X	X								X	
Traceability Analysis						X	X		X				X		X	X
Threat Analysis			X				X		X	X			X			

Testing Special to SQA or V&V	[Fri95]	[Lev95]	[Lyu96]	[Mus98]	[Pf]	[Pr]	[Rak97]	[Rub94]	[Sch98]	[S]	[Voa99]	[Wak99]	[Wal89]
Conformance Test.	X											X	
Configuration Test.						X							
Certification Testing			X	X		X	X		X		X	X	
Reliability Testing	X	X	X	X					X				
Safety Testing	X		X	X					X				
Security Testing					X								
Statistical Testing			X	X	X	X			X	X			
Usability Testing					X			X					
Test Monitoring													X
Test Witnessing													X

Defect Finding Techniques	[Bei90]	[Fen95]	[Fri95]	Hetzel	[Hor96]	[Ipp95]	[Lev95]	[Lew92]	[Lyu96]	[M]	[Mus98]	[Pf]	[Pr]	[Rak97]	[Rub94]	[Sch98]	[S]	[Wak99]	[Wal89]
Algorithm Analysis		X		X				X										X	X
Boundary Value Anal.			X									X	X				X	X	X
Change Impact Anal.									X			X	X	X		X	X		
Checklists					X		X							X					
Consistency Analysis													X			X			
Control Flow Analysis	X	X						X	X			X	X					X	X
Database Analysis	X	X	X					X							X			X	X
Data Flow Analysis	X	X	X					X	X	X					X			X	X
Distrib. Arch. Assess.													X						
Evaluation of Docts.: Concept, Reqmts.			X					X	X						X			X	X

Defect Finding Techniques	[Bei90]	[Fen95]	[Fri95]	Hetzel	[Hor96]	[Ipp95]	[Lev95]	[Lew92]	[Lyu96]	[M]	[Mus98]	[Pf]	[Pr]	[Rak97]	[Rub94]	[Sch98]	[S]	[Wak99]	[Wal89]
Evaluation of Docts.: Design, Code, Test			X					X	X						X			X	
Evaluation of Doc.: User, Installation			X					X	X						X			X	
Event Tree Analysis			X																X
Fault Tree Analysis			X			X			X	X					X				
Graphical Analysis	X	X									X							X	
Hazard Analysis		X	X			X	X		X	X					X				
Interface Analysis	X		X		X			X	X						X				X
Formal Proofs			X						X	X					X				X
Mutation Analysis			X						X									X	X
Perform. Monitoring									X										X
Prototyping			X						X	X					X				X
Reading			X																X
Regression Analysis			X		X			X	X									X	X
Simulation			X																X
Sizing & Timing Anal.			X					X	X	X								X	X
Threat Analysis									X	X					X				

Measurement in Software Quality Analysis	[Bas84]	[Bei90]	[Con86]	[Chi96]	[Fen95]	[Fen97]	[Fri95]	[Gra92]	[Het84]	[Hor96]	[Jon96]	[Kan94]	[Lew92]	[Lyu96]	[Mus89]	[Mus98]	[Pen92]	[Pf]	[Pr]	[McC93]	[Rak97]	[Sch98]	[S]	[Wak99]	[Wei93]	[Zel98]
Benchmarks, profiles, etc.		X				X												X		X		X				
Company Measures Progs.					X	X		X				X						X	X							
Costing		X	X					X		X			X			X		X		X	X	X	X		X	
Customer satisfaction											X	X							X							
Data Collection process	X		X		X	X		X			X															
Debugging		X	X			X								X						X			X			
Defect Analysis		X	X	X			X	X	X	X		X		X		X	X	X	X	X						
Defect Classif. and Descr.		X		X		X	X	X		X		X	X	X		X		X	X							
Defect Features			X	X		X		X		X		X		X							X					
Example of applied GQM						X		X																		
Experimentation:			X	X	X	X										X										X
Framework					X	X																				
GQM	X				X	X		X				X											X			
Methods			X		X			X				X		X		X			X							
Measures			X			X		X		X		X		X			X	X	X		X	X	X			
Models					X	X								X		X										
Prediction						X						X		X		X					X					
Prod. features: O/O Metr.																						X				
Prod. Features: Structure			X		X	X		X						X							X					
Product features: Size			X			X		X				X		X												

Measurement in Software Quality Analysis	[Bas84]	[Bei90]	[Con86]	[Chi96]	[Fen95]	[Fen97]	[Fri95]	[Gra92]	[Het84]	[Hor96]	[Jon96]	[Kan94]	[Lew92]	[Lyu96]	[Mus89]	[Mus98]	[Pen92]	[Pf]	[Pr]	[McC93]	[Rak97]	[Sch98]	[S]	[Wak99]	[Wei93]	[Zel98]
Quality Attributes													X					X	X				X			
Quality Character. Meas.						X							X				X				X					
Reliab. Models & Meas.			X			X	X					X		X		X				X	X					
Scales			X		X	X						X														
SQA & V&V reports *										X						X					X			X		
Statistical tests			X			X								X			X	X					X			
Statistical Analysis & measurement			X		X	X				X		X		X		X	X				X					
Test coverage																X					X					
Theory			X		X	X						X														
Trend analysis														X												
When to stop testing*							X								X	X										

Standards	Quality Requirements & planning	Reviews/ Audits	SQA/V&V planning	Safety/security analysis, tests	Documentation of quality analysis	Measurement
ISO 9000	X	X			X	X
ISO 9126	X					
IEC 61508	X			X		X
ISO/IEC 14598				X	X	X
ISO/IEC 15026	X					
ISO FDIS 15408	X			X		
FIPS 140-1	X			X		
IEEE 730		X	X		X	
IEEE 1008			X			
IEEE 1012		X	X	X	X	
IEEE 1028		X				
IEEE 1228				X		
IEEE 829					X	
IEEE 982.1,.2						X
IEEE 1044						X
IEEE 1061						X

5. Recommended References for Software Quality

5.1. Basic SWEBOK References

Dorfman, M., and R.H. Thayer, *Software Engineering.* IEEE Computer Society Press, 1997. **[D]**

Moore, J.W., *Software Engineering Standards: A User's Road Map.* IEEE Computer Society Press, 1998. **[M]**

Pfleeger, S.L., *Software Engineering – Theory and Practice*. Prentice Hall, 1998. **[Pf]**

Pressman, R.S., *Software Engineering: A Practitioner's Approach* (4th edition). McGraw-Hill, 1997. **[Pr]**

Sommerville, I., *Software Engineering* (5th edition). Addison-Wesley, 1996. **[S]**

5.2. Software Quality KA References

Ackerman, Frank A., "Software Inspections and the Cost Effective Production of Reliable Software," in [D] pp. 235-255. [Ack97]

Basili, Victor R. and David M. Weiss, A Methodology for Collecting Valid Software Engineering Data, IEEE

Transactions on Software Engineering, pp. 728-738, Vol. SE-10, no. 6, November 1984. [Bas84]

Beizer, Boris, *Software Testing Techniques,* International Thomson Press, 1990. [Bei90]

Boehm, B.W. et al., *Characteristics of Software Quality",* TRW series on Software Technologies, Vol. 1, North Holland, 1978. [Boe78]

Chilllarege, Ram, Chap. 9, pp359-400, in [Lyu96]. [Chi96]

Conte, S.D., et al, *Software Engineering Metrics and Models,* The Benjamin / Cummings Publishing Company, Inc., 1986. [Con86]

Ebenau, Robert G., and Susan Strauss, *Software Inspection Process*, McGraw-Hill, 1994. [Ebe94]

Fenton, Norman E., *Software Metrics: A rigorous and practical approach (2nd edition),* International Thomson Computer Press, 1995. [Fen95]

Fenton, Norman E., and Shari Lawrence Pfleeger, Software Metrics, International Thomson Computer Press, 1997. [Fen97]

Freedman, Daniel P., and Gerald M. Weinberg, Handbook of Walkthroughs, Inspections, and Technical Reviews, Little, Brown and Company, 1982. [Fre82]

Friedman, Michael A., and Jeffrey M. Voas, *Software Assessment: reliability, safety testability*, John Wiley & Sons, Inc., 1995. [Fri95]

Grady, Robert B, *Practical Software Metrics for project Management and Process Management,* Prentice Hall, Englewood Cliffs, NJ 07632, 1992. [Gra92]

Hetzel, William, *The Complete Guide to Software Testing,* QED Information Sciences, Inc., 1984, pp177-197. [Het84]

Horch, John W., *Practical Guide to Software Quality Management,* Artech-House Publishers, 1996. [Hor96]

Ippolito, Laura M. and Dolores R. Wallace, NISTIR 5589, A Study on Hazard Analysis in High Integrity Software Standards and Guidelines,@ U.S. Department. of Commerce, Technology Administration, National Institute of Standards and Tech., Jan 1995. http://hissa.nist.gov/HAZARD/ [Ipp95]

Jones, Capers, Applied Software Measurement: Assuring Productivity and Quality, McGraw-Hill, Inc., 2nd edition, 1996.; (Chapters on Mechanics of Measurement and User Satisfaction). [Jon96]

Kan, Stephen, H., *Metrics and Models in Software Quality Engineering*, Addison-Wesley Publishing Co., 1995. [Kan94]

Kazman, R., M. Barbacci, M. Klein, S. J. Carriere, S. G. Woods, Experience with Performing Architecture Tradeoff Analysis, *Proceedings of ICSE 21*, (Los Angeles, CA), IEEE Computer Society, May 1999, 54-63. [Kaz99]

Kiang, David, Harmonization of International Software Standards on Integrity and Dependability, *Proc. IEEE International Software Engineering Standards Symposium*, IEEE Computer Society Press, Los Alamitos, CA, 1995, pp. 98-104. [Kia95]

Laprie, J.C., *Dependability: Basic Concepts and Terminology in English, French, German, Italian and Japanese, IFIP WG 10.4,* Springer-Verlag, New York 1991. [Lap91]

Leveson, Nancy, *SAFEWARE: System Safety and Computers,* Addison-Wesley, 1995. [Lev95]

Lewis, Robert O., *Independent Verification and Validation: A Life Cycle Engineering Process for Quality Software* , John Wiley & Sons, Inc., 1992. [Lew92]

Lyu , Michael R., *Handbook of Software Reliability Engineering*, McGraw Hill, 1996. [Lyu96]

McCall, J.A. - Factors in Software Quality - General Electric, n77C1502, June 1977 [McC77]

McConnell, Steve C., *Code Complete: a practical handbook of software construction,* Microsoft Press, 1993. [McC93]

Musa, John D., and A. Frank Ackerman, "Quantifying Software Validation: When to stop testing?" *IEEE Software*, vol. 6, no. 3, May 1989, 19-27. [Mus89]

Musa, John, *Software Reliability Engineering: More Reliable Software, Faster Development and Testing,* McGraw Hill, 1999. [Mus98]

Peng, Wendy W. and Dolores R. Wallace, "Software Error Analysis," NIST SP 500-209, National Institute of Standards and Technology, Gaithersburg, MD 20899, December 1993.] http://hissa.nist.gov/SWERROR/. [Pen92]

Rakitin, Steven R., *Software Verification and Validation, A Practitioner's Guide*, Artech House, Inc., 1997. [Rak97]

Rubin, Jeffrey, *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests,* John Wiley & Sons, 1994. [Rub94]

Schulmeyer, Gordon C., and James I. McManus, *Handbook of Software Quality Assurance,* Third Edition, Prentice Hall, NJ, 1999. [Sch98]

Voas, Jeffrey, "Certifying Software For High Assurance Environments, " *IEEE Software,* Vol. 16, no. 4, July-August, 1999, pp. 48-54. [Voa99]

Wakid, Shukri, D. Richard Kuhn, and Dolores R. Wallace, "Toward Credible IT Testing and Certification," *IEEE Software,* July-August 1999, 39-47. [Wak99]

Wallace, Dolores R., and Roger U. Fujii, "Software Verification and Validation: An Overview," *IEEE Software*, Vol. 6, no. 3, May 1989, 10-17. [Wal89]

Wallace, Dolores R., Laura Ippolito, and Barbara Cuthill, Reference Information for the Software Verification and Validation Process,@ NIST SP 500-234, NIST, Gaithersburg, MD 20899, April, 1996. http://hissa.nist.gov/VV234/. [Wal96]

Weinberg, Gerald M., Quality Software Management, Vol 2: First-Order Measurement, Dorset House, 1993. (Ch. 8, Measuring Cost and Value). [Wei93]

Zelkowitz, Marvin V. and Dolores R. Wallace, Experimental Models for Validating Technology, *Computer*, Vol. 31 No.5, 1998 pp.23-31. [Zel98]

APPENDIX A – LIST OF FURTHER READINGS

A.1 Books and Articles

Abran, A.; Robillard, P.N. , Function Points Analysis: An Empirical Study of its Measurement Processes, in IEEE Transactions on Software Engineering, vol. 22, 1996, pp. 895-909. [Abr96]

Bevan, N., “Quality and usability: a new framework”, in Achieving Software Product Quality, ed. E. van Veenendaal & J. McMullan, Uitgeverij Tutein Nolthenius, Holland, 1997.[Bev97]

Department of Defense and US Army, Practical Software and Systems Measurement : A Foundation for Objective Project Management, Version 4.0b, October 2000. Available at : www.psmsc.com [DOD00]

Garvin, D., “What Does ‘Product Quality’ Really Mean?” Sloan Management Review, Fall 1984, pp 25-45. [Gar84]

Humphrey, Watts S., Managing the Software Process, Addison Wesley, 1989 Chapters 8, 10, 16. [Hum89]

Hyatt, L.E. and L. Rosenberg, A Software Quality Model and Metrics for Identifying Project Risks and Assessing Software Quality, 8th Annual Software Technology Conference, Utah, April 1996. [Hya96]

Ince, Darrel, *ISO 9001 and Software Quality Assurance*, McGraw-Hill, 1994. [Inc94]

NASA, *Formal Methods Specification and Analysis Guidebook for the Verification of Software and Computer Systems, Volume II: A Practitioner’s Companion*, [NASA-GB-001-97], 1997, http://eis.jpl.nasa.gov/quality/Formal_Methods/. [NAS97]

Palmer, James D., “Traceability,” In: [Dorf], pp. 266-276. [Pal97]

Rosenberg, Linda, Applying and Interpreting Object-Oriented Metrics, Software Tech. Conf. 1998, http://satc.gsfc.nasa.gov/support/index.html. [Ros98]

Vincenti, W.G., What Engineers Know and How They Know It – Analytical Studies form Aeronautical History. Baltimore and London: John Hopkins, 1990. [Vin90]

A.2 Relevant Standards

FIPS 140-1, 1994, Security Requirements for Cryptographic Modules

IEC 61508 Functional Safety - Safety -related Systems Parts 1,2,3

IEEE 610.12-1990, Standard Glossary of Software Engineering Terminology

IEEE 730-1998 Software Quality Assurance Plans

IEEE 829 -1998 Software Test Documentation

IEEE Std 982.1 and 982.2 Standard Dictionary of Measures to Produce Reliable Software

IEEE 1008-1987 Software Unit Test

IEEE 1012-1998 Software Verification and Validation

IEEE 1028 -1997 Software Reviews

IEEE 1044 -1993 Standard Classification for Software Anomalies

IEEE Std 1061-1992 Standard for A Software Quality Metrics Methodology

IEEE Std 1228-1994 Software Safety Plans

ISO 8402-1986 Quality - Vocabulary

ISO 9000-1994 Quality Management and Quality Assurance Standards

ISO 9001-1994 Quality Systems

ISO/IEC 9126-1999: Software Product Quality

ISO 12207 Software Life Cycle Processes 1995

ISO/IEC 12119 Information technology - Software package - Quality requirements and test

ISO/IEC 14598-1998: Software Product Evaluation

ISO/IEC 15026:1998, Information technology -- System and software integrity levels.

ISO/IEC 25939: Information Technology – Software Measurement Process, International Organization for Standardization and the International Electrotechnical Commission, 2000. Available at www.info.uqam.ca/Labo_Recherche/Lrgl/sc7/private_files/07n2410.pdf

The Common Criteria for Information Technology Security Evaluation (CC) VERSION 2.0 / ISO FDIS 15408.