A COMPARISON OF SOFTWARE ASSURANCE METHODS

Marilyn S. Fujii
Logicon, Inc.

ABSTRACT

Several methods are currently employed by software developers to improve software quality. This paper explores the application of three of these methods: quality assurance, acceptance testing, and independent verification and validation. At first glance these methods appear to overlap, but a closer evaluation reveals that each has a distinct objective and an established set of procedures. The purpose of this paper is to clarify the role of each of these methods by examining their scope, organization, and implementation in the software development process.

INTRODUCTION

Over the last decade, there has been a growing concern for software quality. In pursuit of high quality software, researchers and practitioners developed new software engineering concepts, formulated principles for improving the software development process, and implemented several techniques and automated aids intended to discover and eliminate errors from software. All of these efforts have been directed at improving the quality of software and the resulting methods have been described as software assurance methods.[1]

In the broadest sense, a software assurance method is any practice, technique, or tool for improving software quality. This group of methods encompasses much of our existing and emerging software technology. The purpose of this paper is to compare and contrast three widely known (but somewhat misunderstood) software assurance methods: quality assurance, acceptance testing, and independent verification and validation.

The distinction between these three methods often appears cloudy and they may seem to overlap. In reality, there are several distinguishing differences between them, beginning with their objectives. The objective of quality assurance is to define sound software development practices and ensure that they are followed. Acceptance testing's objective is to demonstrate to the customer or user that the developed software's performance is acceptable. Independent verification and validation is performed to ensure that the developed software will not fail its operational mission. It is not surprising that these varied objectives give rise to differing procedures and results for each of the three software assurance methods.

SOFTWARE DEVELOPMENT PROCESS

Quality assurance, acceptance testing, and independent verification and validation each fit within the software development process in a unique way. Software development may be conducted in accordance with several different formal procedures, such as those promulgated by textbooks, guidebooks, and government standards and instructions. Disregarding minor differences in nomenclature and specialized adaptations, the seven basic software development phases illustrated by the rectangles in Figure 1 are common to most well-planned software development efforts.

The first phase of the software development process is the conceptual phase. The software definition effort begins in this phase with feasibility assessments, trade-off studies, and analyses to define specific requirements to be allocated to computer resources. These requirements are defined and documented in a draft system specification which establishes a baseline for the subsequent phases of software development.

The second phase of the software development process is the requirements definition phase. During this phase, system engineering studies determine which system requirements are to be implemented by the software. Analysis determines the software functions needed and the inputs, processing, and outputs required for each function. As part of this phase, the system specification is finalized and a draft software requirements specification is prepared.

Third is the design phase, during which a software design is formulated to realize the functions identified within the software requirements. The design includes the actual algorithms and equations, as well as the control logic and data operations to be performed. During this phase, the software requirements specification is finalized and a draft software design specification is prepared.

In the fourth phase, coding and checkout, the software design is translated into a higher order or assembly level programming language. Compilation and assembly errors are corrected, after which pro-

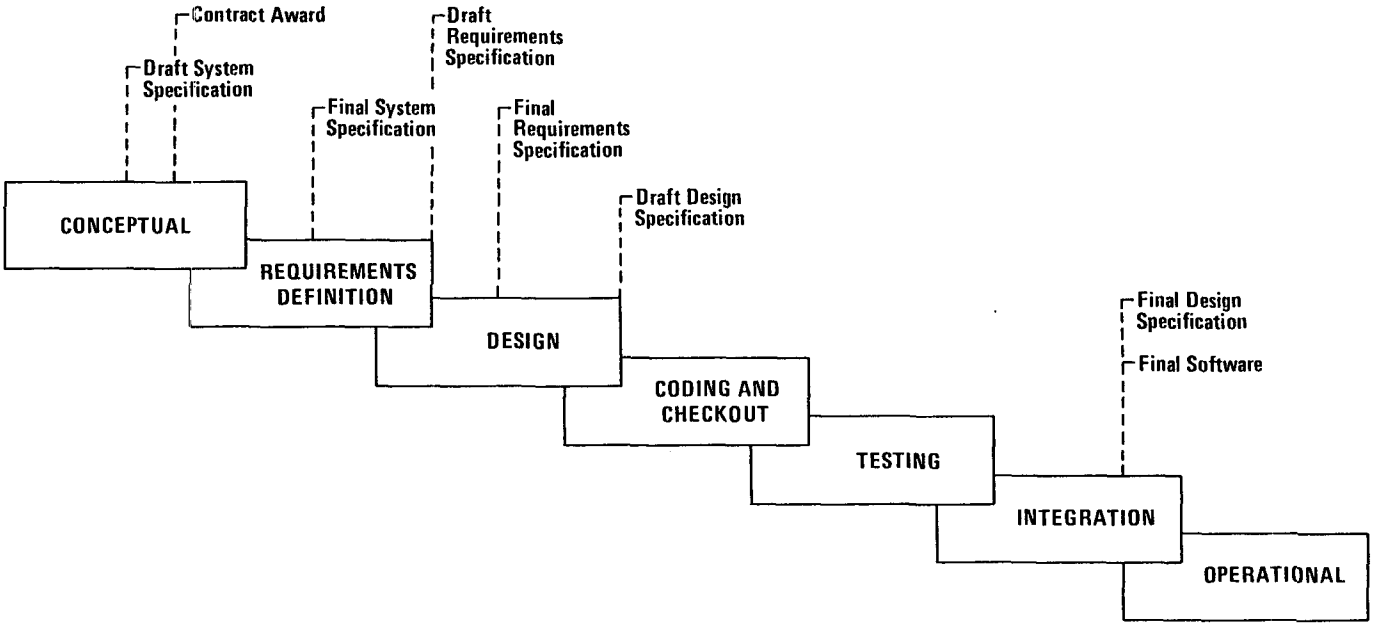

Figure 1. Software Development Process

gram checkout begins by executing individual program modules to remove the more obvious errors. Checkout becomes part of the acceptance testing process at this point and will be discussed in greater detail later.

The next phase is testing. Testing is conducted to show that the developed software's performance is consistent with the established system and software requirements. Testing is a major element of the acceptance testing process, hence will be discussed in greater detail later.

The integration phase brings together all of the system components, hardware as well as software. System and operational testing is conducted to assure the satisfaction of the system requirements in the actual or simulated system environment. Near the completion of the integration phase, the software design specification is updated to reflect the "as built" software design and the final software is delivered.

The final software development phase is the operational phase, during which the software must be maintained. At this point, the software has been accepted for operational use and further activity consists of modifying the program to remove latent errors, or in response to new or revised requirements.[2]

ROLE OF ASSURANCE METHODS

Quality assurance, acceptance testing, and independent verification and validation are each unique in their relationship to the software development process. Quality assurance activities represent a low level of effort, focusing on audits for completeness and compliance with software development procedures. Acceptance testing is actually a subset of software development activities consisting of the entire testing phase and portions of other software development phases. Independent verification and validation applies an intermediate level of effort in parallel with software development to evaluate the software as it evolves.

The following paragraphs discuss each software assurance method in greater detail.

Quality Assurance

Quality assurance consists of defining valid software development standards and monitoring the software products to ensure that they comply with these standards. Unfortunately for software developers, although quality assurance principles and procedures are well understood for hardware, their extension to software is not straightforward and is comparatively undocumented. As an illustration of this point, the standard reference on quality assurance does not even address the problem of software assurance.[3]

Standards for software quality assurance have primarily been established by large software houses and government agencies. These standards can be incorporated into a contract by reference, directing the contractor to develop and implement a quality assurance plan for all deliverable software.

The quality assurance plan specifies the organization and scope of the quality assurance program. Quality assurance should be organized as a separate function to centralize the requisite expertise in a quality assurance group and to give this group sufficient autonomy and clout to implement a truly effective program. In scope, the quality assurance program continues on a low level of effort throughout the software development process.

The activities that comprise the quality assurance effort are shown in Figure 2. Early activity focuses on defining quality assurance procedures and standards in a formal plan and supporting configuration management and computer program development planning. The remainder of the effort is devoted to reviewing and auditing software products, as they are developed, against the standards established in the plan. Quality assurance conducts all design reviews and audits that regulate the software development. They evaluate all draft and final documentation, such as requirement and design specifications, test plans and procedures, and user manuals. They also schedule and conduct code walk-throughs and witness acceptance testing. In preparation for delivery fo the final software, they audit the final software configuration to be installed in the operational environment.[4]

When properly implemented as a software assurance method, quality assurance can establish the following results:

- Adherance to coding standards and conventions
- Compliance with documentation requirements and standards
- Successful completion of activities according to criteria specified in the quality assurance plan

Acceptance Testing

Acceptance testing is the established means for demonstrating to the customer or user that the developed software is acceptable for operational use. Because of its complexity, software does not lend itself to exhaustive testing and the adequacy of current test methodology has become an extremely important issue. Acceptance testing, as defined in most standards, stresses a functional testing approach that combines the testing of individual software functions with performance testing for the entire program.

A subset of the software development project staff plans and conducts acceptance testing, perhaps with some consultation from the quality assurance organization. This gives rise to a potential management problem that must be averted. When staffing the acceptance testing effort, the desire to assign the most talented, experienced personnel to design and development may inadvertantly relegate acceptance testing to a secondary role, thereby lessening its effectiveness. It is also difficult to prevent time or economic pressures from limiting the scope of acceptance testing.

Acceptance testing consists of three activities as depicted in Figure 3. The first activity, test planning, determines from the software requirements what tests should be performed for each software function and what tests should exercise the entire computer program. A test plan is developed from these findings and test procedures are prepared to specify the actual tests in detail. Next, preliminary qualification testing is conducted to establish the proper execution of each software function. Last, formal qualification testing is performed to demonstrate that the integrated software operates correctly in the user environment.

Acceptance testing, with proper emphasis and management, can provide the following results:

- Operation of each software function under nominal conditions

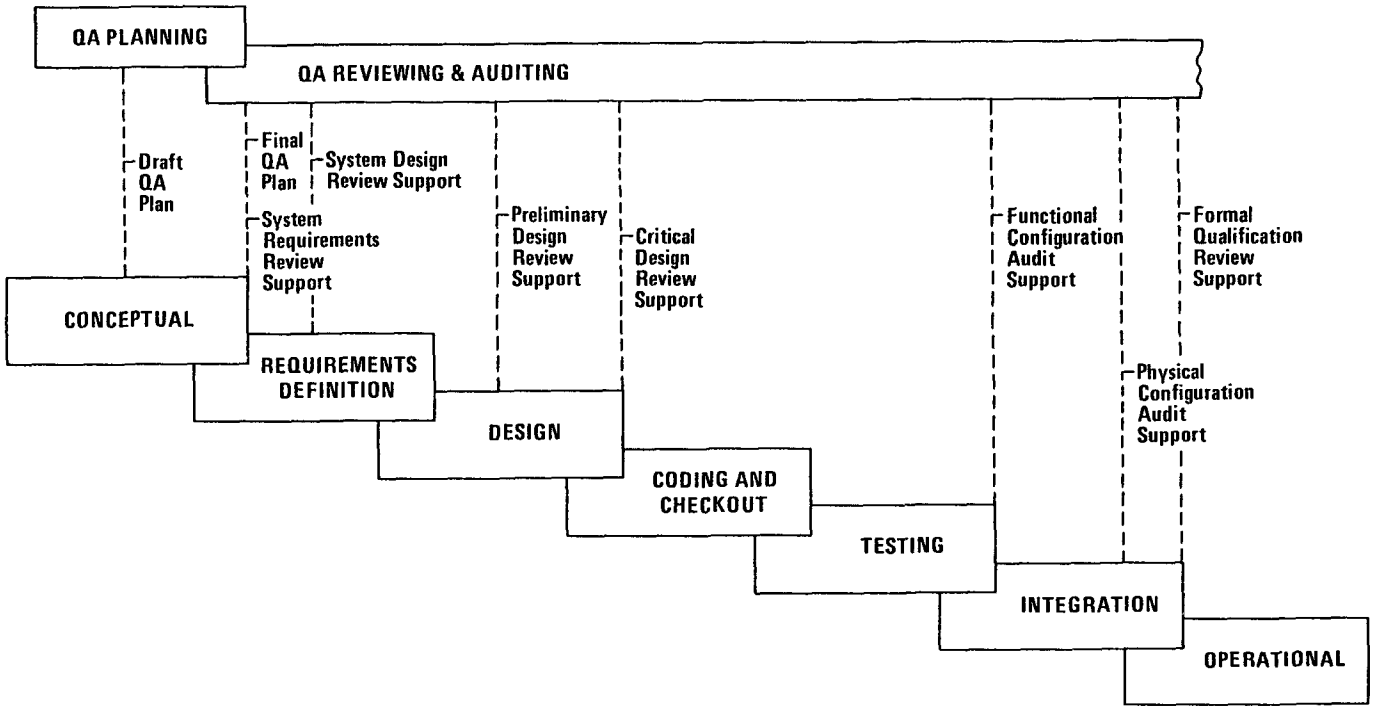

Figure 2. Quality Assurance Activities

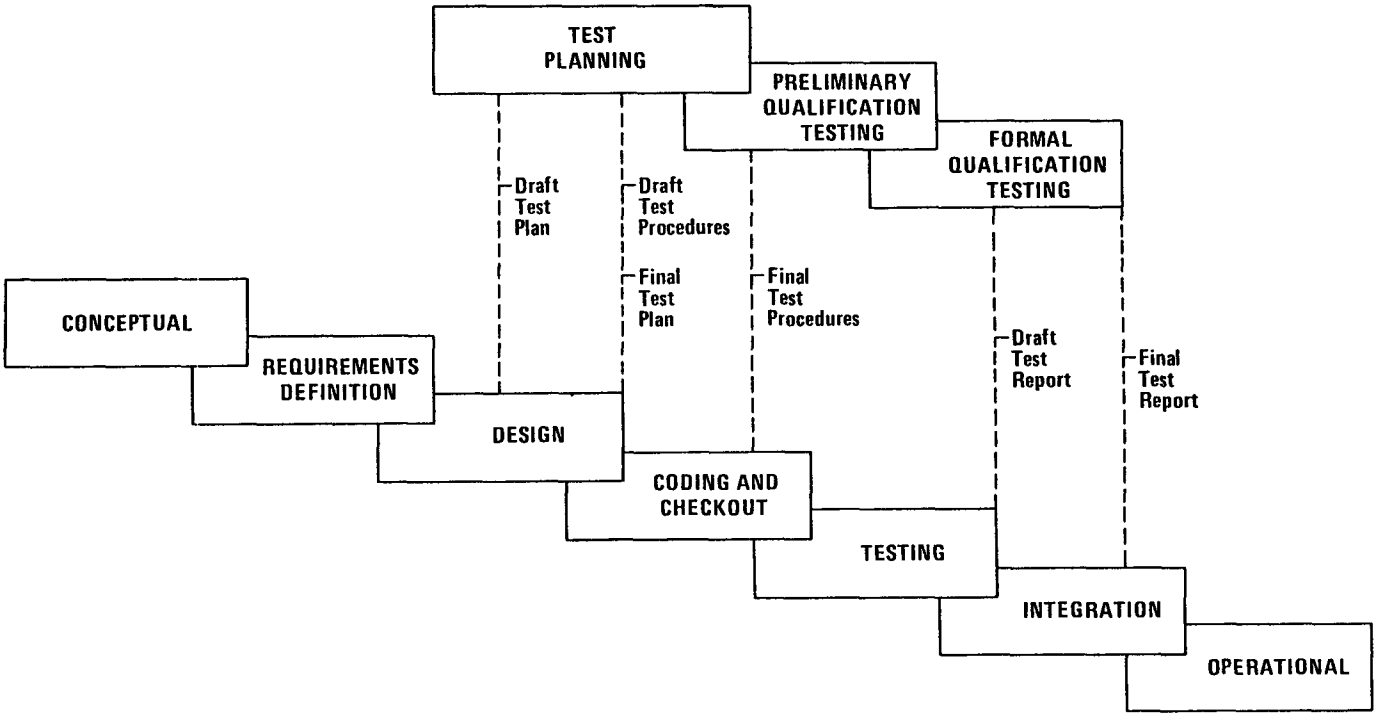

Figure 3. Acceptance Testing Activities

- Operation of the integrated program under nominal conditions
- Compliance of the program with general performance requirements
- Discovery of most obvious errors

Independent Verification and Validation

Independent verification and validation is a software assurance method employed to achieve the high reliability requirements of critical software, such as command and control, air traffic control, weapon system, mission planning, or communications software. The objectives of independent verification and validation are to ensure that the software will not fail in its operational mission, either by failing to correctly perform an intended function or by unintentionally performing an undesirable function.

As implied by its name, independent verification and validation is performed by a group that is organizationally independent of the software development project staff. Independence provides a fresh viewpoint needed to accurately assess the software and it also precludes bias in critiquing the software products. The independent verification and validation effort is performed in parallel with the software development so that errors detected can be corrected in a cost-effective manner. The goal is to eliminate all critical errors from the software prior to its operational use.

There are five major activities that comprise independent verification and validation, as shown in Figure 4. The first, independent verification and validation planning, schedules the entire effort so that it synchronizes with the development to provide results when needed at key decision points. Next, requirements analysis checks the software requirements for completeness, correctness, consistency, traceability, and testability. The third activity, design analysis, evaluates the logical and mathematical design for correctness and for satisfaction of the software requirements. Code analysis examines the source language and object code to verify that the software is a correct implementation of the design. Independent testing evaluates the developed software's performance under both nominal and worst-case scenarios, with particular emphasis on discovering subtle errors that would not easily be revealed during acceptance testing.[5]

The aim of independent verification and validation is to find potential causes of catastrophic failures in critical software. As a software assurance method, independent verification and validation can provide the following results:

- Identification of serious requirement and design errors that would otherwise remain undetected until acceptance testing
- Identification of latent or conditional errors that would otherwise remain undetected
- Assurance that user requirements have not been overlooked
- Assurance that critical functions will not fail in operational use

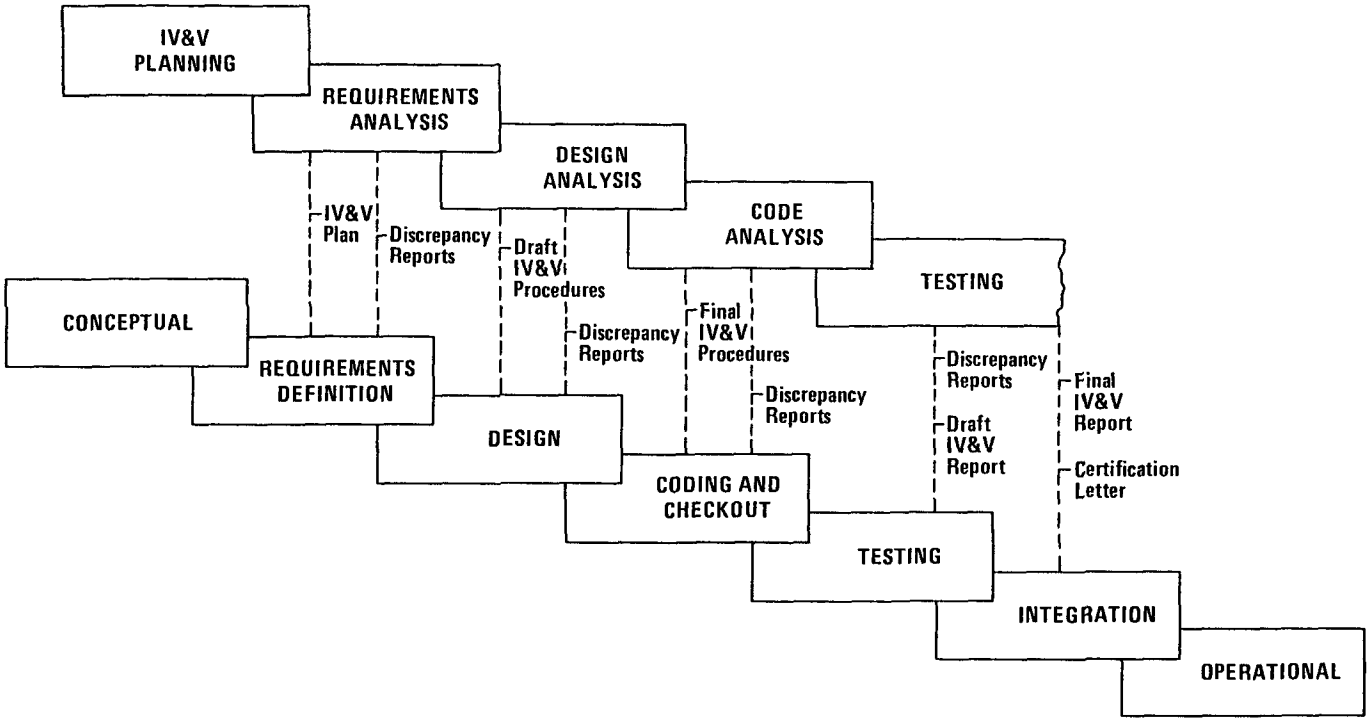

Figure 4. Independent Verification and Validation Activities

Table 1. Comparison of Software Assurance Methods

Factor	Quality Assurance	Acceptance Testing	Independent V&V
Primary Objective	Enforce Standards	Demonstrate Acceptable Performance	Eliminate Critical Errors
Organization	Independent Group	Development Group	Independent Group
Relative Size	Small Staff, Long Duration	Large Staff, Medium Duration	Medium Staff, Long Duration
Applicability	Deliverable Software	Operational Software	Critical Software
Major Strength	Cost-Benefit Ratio	Widely Established	Extremely Effective
Major Weakness	Difficult to Initiate	Subject to Tunnel Vision	Requires Additional Resources

COMPARING AND CONTRASTING

A closer look at quality assurance, acceptance testing, and independent verification and validation has revealed that they are indeed three separate software assurance methods. A comparison of these methods, summarized in Table 1, shows that that their differences begin with their objectives.

In further contrast, quality assurance is performed by a small, independent group that enforces standards for deliverable software throughout the software development effort. Acceptance testing comprises a significant portion of the development group's effort to check out, test, and integrate operational software. Independent verification and validation is performed by a medium-sized, independent organization that evaluates critical software in parallel with its development.

Each software assurance method has major strengths and weaknesses. Quality assurance has a very favorable cost-benefit ratio -- it improves software while not requiring a great investment in manpower and specialized tools. There is some initial difficulty, however, in establishing a quality assurance group and developing the appropriate standards.

Acceptance testing is very widely established and well understood. Nevertheless, there is the com-

pulsion to short-change acceptance testing if project pressures mount and a tendency to develop tunnel vision that prevents discovery of errors.

Independent verification and validation has been shown to be an extremely effective method for detecting errors that would otherwise cause catastrophic system failures. This method requires the investment of additional resources which must be justified on the basis of life-cycle costs or risk avoidance.

CONCLUSION

Quality assurance, acceptance testing, and independent verification and validation each play complementary roles in software assurance. An integrated approach for assuring the development of high quality software will recognize the criteria for applying each method and employ each as it is intended, rather than substituting one method for another.

REFERENCES

1. Wasserman, Anthony I., L. A. Belady, et al., "Software Engineering: The Turning Point," Computer, September 1978.

2. "An Overview of Software Development and Management," Management Guide to Software Acquisition, Aeronautical Systems Division of Air Force Systems Command, ASD-TR-76-11, Volume I, June 1976.

3. Juran, J. M. et al., Quality Control Handbook, Third Edition, McGraw-Hill Book Company, 1974.

4. Foster, Richard A., Introduction to Software Quality Assurance, Third Edition, 1975.

5. Fujii, Marilyn S., "Independent Verification of Highly Reliable Programs," Proceedings of the Computer Software and Applications Conference, 8-11 November 1977.