A Overview of Software Verification & Validation and Selection Process

Omprakash Deshmukh
M.Tech.(SE) IIIT Gwalior
System Analyst, Netcracker
#5&43 HiTec City,Hyderabad, AP - 500081.

Mandakini Kaushik,
M.Tech.(CSE) Scholar, Dept. of CSE,,
Rungta College of Engg. & Tech.,
Bhilai – 490 024(C.G.), INDIA

***Abstract*—** In recent years many Software verification and validation techniques are introduced. This curriculum paper provides an overview needed to understand in-depth curriculum modules in the verification and validation area. This paper also addresses planning considerations for V&V processes, including the selection and integration of V&V techniques throughout the software evolution process. Our work identified, along with their possible V&V objectives. The V&V process consists of numerous techniques and tools, often used in combination with one another. Due to the large number of V&V approaches in use, this paper cannot address every technique. Instead, it will analyze five categories of V&V approaches. These are:

• Technical reviews,
• Software testing,
• Proof of correctness (program verification),
• Simulation and prototyping, and
• Requirements tracing.

In the development of a software system, it is important to be able to determine if the system meets specifications and if its outputs are correct. This is the process of verification and validation (V & V) and its planning must start early in the development life cycle. Both aspects are necessary as a system meeting its specifications does not necessary mean it is technically correct and vice versa. There are many different V & V techniques which are applicable at different stages of the development life cycle. Thorough V & V does not prove that the system is safe or dependable, and there is always a limit to how much testing is enough testing. Therefore, extreme care should be taken in the development of software systems to make sure that the right amount of time is spent on V & V [7].

Software quality is achieved through the application of development techniques and the use of verification procedures throughout the development process Careful consideration of specific quality attributes and validation requirements leads to the selection of a balanced collection of review, analysis, and testing techniques for use throughout the life cycle. This paper surveys current verification, validation, and testing approaches and discusses their strengths, weaknesses, and life-cycle usage. In conjunction with these, the Paper describes automated tools used to implement validation, verification, and testing. In the discussion of new research thrusts, emphasis is given to the continued need to develop a stronger theoretical basis for testing and the need to employ combinations of tools and techniques that may vary over each application [7].

Keywords- verification and validation, system Testing, Module Testing, Regreation Testing, simulation and prototyping, requirement tracing, Proof of correctness

I. INTRODUCTION

Programming is an exercise in problem Solving. As with any problem-solving activity, determination of the validity of the solution is part of the process. This survey discusses testing and analysis techniques that can be used to validate software and to instill confidence in the quality of the programming product. It presents a collection of verification techniques that can be used throughout the development process to facilitate software quality assurance.

1. Terminology

The evolution of software that satisfies its user expectations is a necessary goal of a successful soft ware development organization. To achieve this goal, software engineering practices must be applied throughout the evolution of the software product. Most of these software engineering practices attempt to create and modify software in a manner that maximizes the probability of satisfying its user expectations. Other practices, addressed in this module, actually attempt to insure that the product will meet these user expectations. These practices are collectively referred to as software verification and validation (V&V). The reader is cautioned that terminology in this area is often confusing and conflicting. The glossary of this module contains complete definitions of many of the terms often used to discuss V&V practices [7]. This section attempts to clarify terminology as it will be used in the remainder of the module.

- Validation refers to the process of evaluating soft- ware at the end of its development to insure that it is free from failures and complies with its requirements. A failure is defined as incorrect product behavior. Often this validation occurs through the utilization of various testing approaches. Other intermediate software products may also be validated, such as the validation of a requirements description through the utilization of a prototype.
- Verification refers to the process of determining whether or not the products of a given phase of a software development process fulfill the requirements established during the previous phase. Software technical reviews represent one common approach for verifying various products. For example, a specifications review will normally attempt to verify the specifications description against a requirements description. Proof of correctness is another technique for verifying programs to formal specifications. Verification approaches attempt to identify product faults or errors, which give rise to failures.

2. Evolving Nature of Area

As the complexity and diversity of software products continue to increase, the challenge to develop new and more effective V&V strategies continues. The V&V approaches that were reasonably effective on small batch-oriented products are not sufficient for concurrent, distributed, or embedded products. Thus, this area will continue to evolve as new research results emerge in response to new V&V challenges.

II. OVERVIEW OF V&V IN SOFTWARE EVOLUTION

The evolution of a software product can proceed in many ways, depending upon the development approach used. The development approach determines the specific intermediate products to be created. For any given project, V&V objectives must be identified for each of the products created.

1. Types of Products

To simplify the discussion of V&V objectives, five types of products are considered in this module. These types are not meant to be a partitioning of all software documents and will not be rigorously defined. Within each product type, many different representational forms are possible. Each representational form determines, to a large extent, the applicability of particular V&V approaches. The intent here is not to identify V&V approaches applicable to all products in any form, but instead to describe V&V approaches for representative forms of products. References are provided to other sources that treat particular approaches in depth.

- Requirements: The requirements document [9]:"customer/user-oriented requirements" or C-requirements) provides an informal statement of the user's needs.
- Specifications: The specifications document (Rombach: "design- oriented requirements" or D-requirements) provides a refinement of the user's needs, which must be satisfied by the product. There are many approaches for representing specifications, both formal and informal [9]
- Designs: The product design describes how the specifications will be satisfied. Depending upon the development approach applied in the project, there many design representation approaches are described in Introduction to Software Design [10].
- Implementations: "Implementation" normally refers to the source code for the product. It can, however, refer to other implementation-level products, such as decision tables [11].
- Changes: Changes describe modifications made to the product. Modifications are normally the result of error corrections or additions of new capabilities to the product.

2. V&V Objectives

The specific V&V objectives for each product must be determined on a project-by-project basis. This determination will be influenced by the criticality of the product, its constraints, and its complexity. In sure that the product satisfies the user needs. Thus, everything in the product's requirements and specifications must be the target of some V&V activity. In order to limit the scope of this module, however, the V&V approaches described will concentrate on the for many systems, will not be addressed here. This is consistent with the V&V approaches normally described in the literature. The broader picture of "assurance of software quality" is addressed else- where [12].

Limiting the scope of the V&V activities to functionality and performance, five general V&V objectives can be identified [3]. These objectives provide a framework within which it is possible to determine the applicability of various V&V approaches and techniques [7].

- Correctness: The extent to which the product is fault free.
- Consistency: The extent to which the product is consistent within itself and with other products.
- Necessity: The extent to which everything in the product is necessary.
- Sufficiency: The extent to which the product is complete.
- Performance: The extent to which the product satisfies its performance requirements.

III. SOFTWARE V&V APPROACHES AND THEIR APPLICABILITY

Software V&V activities occur throughout the evolution of the product. There are numerous techniques and tools that

may be used in isolation or in combination with each other. In an effort to organize these V&V activities, five broad classifications of approaches are presented. These categories are not meant to provide a partitioning, since there are some techniques that span categories. Instead, the categories represent a practical view that reflects the way most of the V&V approaches are described in the literature and used in practice. Possible combinations of these approaches are discussed in the next section.

1. Software Technical Reviews

The software technical review process includes techniques such as walk-through, inspections, and audits. Most of these approaches involve a group meeting to assess a work product. A comprehensive examination of the technical review process and its effectiveness for software products is presented in The Software Technical Review Process.
Software technical reviews can be used to examine all the products of the software evolution process. In particular, they are especially applicable and necessary for those products not yet in machine process able form, such as requirements or specifications written in natural language.

2. Software Testing

Software testing is the process of exercising a product to verify that it satisfies specified requirements or to identify differences between expected and actual results [3].

- Levels of Testing: In this section, various levels of testing activities, each with its own specific goals, are identified and described. This listing of levels is not meant to be complete, but will illustrate the notion of levels of testing with particular goals. Other possible levels of testing not addressed here include acceptance testing, alpha testing, beta testing, etc. [4].

(i) Module Testing

Module (or unit) testing is the lowest level of testing and involves the testing of a software module or unit. The goal of module-level testing is to insure that the component being tested conforms to its specifications and is ready to be integrated with other components of the product. Module testing is treated in depth in the curriculum module Unit Testing and Analysis

(ii) Integration Testing

Integration testing consists of the systematic combination and execution of product components. Multiple levels of integration testing are possible with a combination of hardware and software components at several different levels. The goal of integration testing is to insure that the interfaces between the components are correct and that the product components combine to execute the product's functionality correctly.

(iii) System Testing

System testing is the process of testing the integrated hardware and software system to verify that the system meets its specified requirements [3]. Practical priorities must be established to complete this task effectively. One general priority is that system testing must concentrate more on system capabilities rather than component capabilities [4]. This suggests that system tests concentrate on insuring the use and interaction of functions rather than testing the details of their implementations. Another priority is that testing typical situations is more important that testing special cases. This suggests that test cases be constructed corresponding to high-probability user scenarios. This facilitates early detection of critical problems that would greatly disrupt a user.

(iv) Regression Testing

Regression testing can be defined as the process of executing previously defined test cases on a modified program to assure that the software changes have not adversely affected the program's previously existing functions. The error-prone nature of software modification demands that regression testing be performed. An important regression testing strategy is to place a higher priority on testing the older capabilities of the product than on testing the new capabilities provided by the modification [13]. This insures that capabilities the user has become dependent upon are still intact. This is especially important when we consider that a recent study found that half of all failures detected by users after a modification were failures of old capabilities, as a result of side effects of implementation of new functionality. The effectiveness of these strategies is highly dependent upon the utilization of test matrices (see below), which enable identification of coverage provided by particular test cases.

IV. Testing Techniques and their Applicability

(i) Functional Testing and Analysis

Functional testing develops test data based upon documents specifying the behavior of the software. The goal of functional testing is to exercise each aspect of the software's specified behavior over some subset of its input. How- den has developed an integrated approach to testing based upon this notion of testing each aspect of specified behavior [7]. A classification of functional testing approaches and a description of representative techniques is presented in [14]. Functional testing and analysis techniques are applicable for all levels of testing. However, the level of specified behavior to be tested will normally be at a higher level for

integration and system-level testing. Thus, at a module level, it is appropriate to test boundary conditions and low-level functions, such as the correct production of a particular type of error message.

Limitation: The automation of functional testing techniques has been hampered by the informality of commonly used specification techniques. The difficulty lies in the identification of the functions to be tested. Some limited success in automating this process has been obtained for some more rigorous specification techniques.

(ii) Structural Testing and Analysis

Structural testing develops test data based upon the implementation of the product. Usually this testing occurs on source code. However, it is possible to do structural testing on other representations of the program's logic. Structural testing and analysis techniques include data flow anomaly detection, data flow coverage assessment, and various levels of path b. Testing Techniques and their Applicability coverage. A classification of structural testing approaches and a description of representative techniques is presented in [14]

Structural testing and analysis are applicable to module testing, integration testing, and regression testing. At the system test level, structural testing is normally not applicable, due to the size of the system to be tested. For example, appear discussing the analysis of a product consisting of 1.8 million lines of code, suggests that over 250,000 test cases would be needed to satisfy coverage criteria [2]. At the module level, all of the structural techniques are applicable. As the level of testing increases to the integration level, the focus of the structural techniques is on the area of interface analysis [6]. This interface analysis may involve module interfaces, as well as interfaces to other system components. Structural testing and analysis can also be performed on designs using manual walk-through or design simulations [3].

Limitation: Structural testing and analysis techniques are very effective in detecting failures during the module and integration testing levels. Structural testing is very cumbersome to perform without tools, and even with tools requires considerable effort to achieve desirable levels of coverage. Since structural testing and analysis techniques cannot detect missing functions (nor some other types of errors), they must be used in combination with other strategies to improve failure detection effectiveness

(iii) Error-Oriented Testing and Analysis

Error-oriented testing and analysis techniques are those that focus on the presence or absence of errors in the programming process.

Error-oriented testing and analysis techniques are, in general, applicable to all levels of testing. Some techniques, such as statistical methods, error seeding, and mutation testing, are particularly suited to application during the integration and system levels of testing

(iv) Hybrid Approaches

Combinations of the functional, structural, and error-oriented techniques have been investigated and are described in [14]. These hybrid approaches involve integration of techniques, rather than their composition. Hybrid approaches, particularly those involving structural testing, are normally applicable at the module level.

(v) Integration Strategies

Integration consists of the systematic combination and analysis of product components. It is assumed that the components being integrated have already been individually examined for correctness. This insures that the emphasis of the integration activity is on examining the interaction of the components [4, 6]. Although integration strategies are normally discussed for implementations, they are also applicable for integrating the components of any product, such as designs.

(vi) Transaction Flow Analysis

Transaction flow analysis develops test data to execute sequences of tasks that correspond to transaction, where a "transaction" is defined as a unit of work seen from a system user's point of view [4, 1, and 2]. An example of a transaction for an operating system might be a request to print a file. The execution of this transaction requires several tasks, such as checking the existence of the file, validating permission to read the file, etc.

The first step of transaction flow analysis is to identify the transactions. McCabe suggests the drawing of data flow diagrams after integration testing to model the logical flow of the system. Each transaction can then be identified as a path through the data flow diagram, with each data flow process corresponding to a task that must be tested in combination with other tasks on the transaction flow [1]. Once the transaction flows have been identified, black-box testing techniques can be utilized to generate test data for selected paths through the transaction flow diagram.

Transaction flow analysis is a very effective technique for identifying errors corresponding to interface problems with functional tasks. It is most applicable to integration and system level testing. The technique is also appropriate for addressing completeness and correctness issues for requirements, specifications, and designs.

(vii) Stress Analysis

Stress analysis involves analyzing the behavior of the system when its resources are saturated, in order to assess whether or not the system will continue to satisfy its specifications. For example, one typical stress test for an

operating system would be a program that requests as much memory as the system has available.

The first step in performing a stress analysis is identifying those resources that can and should be stressed. This identification is very system dependent, but often includes resources such as file space, memory, I/O buffers, processing time, and interrupt handlers. Once these resources have been identified, test cases must be designed to stress them. These tests often require large amounts of data, for which automated support in the form of test-case generators is needed.

Although stress analysis is often viewed as one of the last tasks to be performed during system testing, it is most effective if it is applied during each of the product's V&V activities. Many of the errors detected during a stress analysis correspond to serious design flaws. For example, a stress analysis of a design may involve an identification of potential bottlenecks that may prevent the product from satisfying its specifications under extreme loads

Stress analysis is a necessary complement to the previously described testing and analysis techniques for resource-critical applications. Stress analysis techniques can also be combined with other approaches during V&V activities to insure that the product's specifications for such attributes as performance, safety, security, etc., are met.

(viii) Failure Analysis

Failure analysis is the examination of the product's reaction to failures of hardware or software. The product's specifications must be determined precisely which types of failures must be analyzed and what the product's reaction must be. Failure analysis is sometimes referred to as "recovery testing" Failure analysis must be performed during each of the product's V&V activities. It is essential during requirement and specification V&V activities that a clear statement of the product's response to various types of failures be addressed in terms that allow analysis. The design must also be analyzed to show that the product's reaction to failures satisfies its specifications. The failure analysis of implementations often occurs during system testing. This testing may take the form of simulating hardware or software errors or actual introduction of these types of errors.

Failure analysis is essential to detecting product recovery errors. These errors can lead to lost files, lost data, duplicate transactions, etc. Failure analysis techniques can also be combined with other approaches during V&V activities to insure that the product's specifications for such attributes as performance, security, safety, usability, etc., are met.

(ix) Concurrency Analysis

Concurrency analysis examines the interaction of tasks being executed simultaneously within the product to insure that the overall specifications are being met. Concurrent tasks may be executed in parallel or have their execution interleaved. Concurrency analysis is sometimes referred to as "background testing". For products with tasks that may execute in parallel, concurrency analysis must be performed during each of the product's V&V activities. During design, concurrency analysis should be performed to identify such issues as potential contention for resources, deadlock, and priorities. A concurrency analysis for implementations normally takes place during system testing. Tests must be designed, executed, and analyzed to exploit the parallelism in the system and insure that the specifications are met.

(x) Performance Analysis

The goal of performance analysis is to insure that the product meets its specified performance objectives. These objectives must be stated in measurable terms, so far as possible. Typical performance objectives relate to response time and system throughput [4].

A performance analysis should be applied during each of the product's V&V activities. During requirement and specification V&V activities, performance objectives must be analyzed to insure completeness, feasibility, and testability. Prototyping, simulation, or other modeling approaches may be used to insure feasibility. For designs, the performance requirements must be allocated to individual components. These components can then be analyzed to determine if the performance requirements can be met. Prototyping, simulation, and other modeling approaches again are techniques applicable to this task. For implementations, a performance analysis can take place during each level of testing. Test data must be carefully constructed to correspond to the scenarios for which the performance requirements were specified.

V. ANOTHER V&V TECHNIQUES

- Proof of Correctness: Proof of correctness is a collection of techniques that apply the formality and rigor of mathematics to the task of proving the consistency between an algorithmic solution and a rigorous, complete specification of the intent of the solution. This technique is also often referred to as "formal verification."

Limitation: There are several limitations to proof of correctness techniques. One limitation is the dependence of the technique upon a correct formal specification that reflects the user's needs. Current specification approaches cannot always capture these needs in a formal way, especially when product aspects such as performance, reliability, quality, etc., are considered. Another limitation has to do with the complexity of rigorously specifying the execution behavior of the computing environment.

- Simulation and Prototyping: Simulation and prototyping are techniques for analyzing the expected behavior of a product. For V&V purposes, simulations and prototypes are normally used to analyze requirements and specifications to insure that they reflect the user's needs. Simulations and prototypes can also be used to analyze

predicted product performance, especially for candidate product designs, to insure that they conform to the requirements. It is important to note that the utilization of simulation and prototyping as V&V techniques requires that the simulations and prototypes themselves be correct. Thus, the utilization of these techniques requires an additional level of V&V activity.

- Requirements Tracing: Requirements tracing is a technique for insuring that the product, as well as the testing of the product, addresses each of its requirements. The usual approach to performing requirements tracing uses matrices. One type of matrix maps requirements to software modules. Construction and analysis of this matrix can help insure that all requirements are properly addressed by the product and that the product does not have any superfluous capabilities Requirements tracing can be applied for all of the products of the software evolution process.

VI. IDENTIFICATION OF V&V GOAL

V&V goals must be identified from the requirements and specifications. These goals must address those attributes of the product that correspond to its user expectations. These goals must be achievable, taking into account both theoretical and practical limitations

VII. SELECTION OF V&V TECHNIQUESDENTIFICATION

Once a set of V&V objectives has been identified, specific techniques must be selected for each of the project's evolving products. We see the selection of V&V techniques based on software life cycle.

- Requirements: The applicable techniques for accomplishing the V&V objectives for requirements are technical reviews, prototyping, and simulation. The review process is often called a System Requirements Review (SRR). Depending upon the representation of the requirements, consistency analyzers may be used to support the SRR.
- Specifications: The applicable techniques for accomplishing the V&V objectives for specifications are technical reviews, requirements tracing, prototyping, and simulation. The specifications review is sometimes combined with a review of the product's high-level design. The requirements must be traced to the specifications.
- Designs: The applicable techniques for accomplishing the V&V objectives for designs are technical reviews, requirements tracing, prototyping, simulation, and proof of correctness. High-level designs that correspond to an architectural view of the product are often reviewed in a Preliminary Design Review. Detailed designs are addressed by a Critical Design Review. Depending upon the representation of the design, static analyzers may be used to assist these review processes. Requirements must be traced to modules in the architectural design; matrices can be used to facilitate this process [16]. Prototyping and simulation can be used to assess feasibility and adherence to performance requirements. Proofs of correctness, where applicable, are normally performed at the detailed design level
- Implementations: The applicable techniques for accomplishing the V&V objectives for implementations are technical reviews, requirements tracing, testing, and proof of correctness. Various code review techniques such as walk-through and inspections exist. At the source-code level, several static analysis techniques are available for detecting implementation errors. The requirements tracing activity is here concerned with tracing requirements to source-code modules. The bulk of the V&V activity for source code consists of testing. Multiple levels of testing are usually performed. Where applicable, proof-of-correctness techniques may be applied, usually at the module level.
- Changes: Since changes describe modifications to products, the same techniques used for V&V during development may be applied during modification. Changes to implementations require regression testing.

VIII. V&V LIMITATION

The overall objective of software V&V approaches is to insure that the product is free from failures and meets its user's expectations. There are several theoretical and practical limitations that make this objective impossible to obtain for many products.

- Theoretical Foundations:Some of the initial theoretical foundations for testing were presented by Goodenough and Gerhart in their classic paper. This paper provides definitions for reliability and validity, in an attempt To characterize the properties of a test selection strategy. A mathematical framework for investigating testing that enables comparisons of the power of testing methods is described in [15]. How-den claims the most important theoretical result in program testing and analysis is that no general purpose testing or analysis procedure can be used to- prove program correctness. A proof of this result is contained in his text [6].
- Impracticality of Testing All Data: For most programs, it is impractical to attempt to test the program with all possible inputs, due to a combinatorial explosion. For those inputs selected, a testing oracle is needed to be-determine the correctness of the output for a particular test input.
- Impracticality of Testing All Paths: For most programs, it is impractical to attempt to test all execution paths through the product, due to a combinatorial explosion. It is also not possible to develop an algorithm for generating

test- data for paths in an arbitrary product, due to the inability to determine path feasibility.

- No Absolute Proof of Correctness: Howden claims that there is no such thing as an absolute proof of correctness. Instead, he suggests that there are proofs of equivalency, i.e., proofs that one description of a product is equivalent to another description. Hence, unless a formal specification can be shown to be correct and, indeed, reflects exactly the user's expectations, no claim of product correctness can be made.

IX. CONCLUSION

In conclusion, verification and validation is a crucial part of the development life cycle of an software system. Verification starts from the requirements analysis stage where design reviews and checklists are used to the validation stage where functional testing and environmental modeling is done. V & V is very important and an issue that comes up is how much verification is enough verification. Obviously, the more testing the better, but when do the benefits from testing outweigh the cost and time of the project. This will vary from project to project and only the developer can determine this. In addition, V & V cannot be used to prove that a system is safe or dependable [3].

Software testing is an important technique for the improvement and measurement of a software system quality. But it is really not possible to find out all the errors in the program. So, the fundamental question arises, which strategy we would adopt to test. In my paper, I have described some of the most prevalent and commonly used strategies of software testing which are classified by purpose and they are classified into [5]

- Correctness testing, which is used to test the right behavior of the system and it is further divided into black box, white box and grey box testing techniques (combines the features of black box and white box testing).
- Performance testing, which is an independent discipline and involves all the phases as the main stream testing life cycle i.e. strategy, plan, design, execution, analysis and reporting. Performance testing is further divided into load testing and stress testing.
- Reliability testing, which discovers all the failure of the system and removes them before the system deployed.
- Security testing makes sure that only the authorized personnel can access the system and is further divided into Security Auditing and Scanning, Vulnerability Scanning, Risk Assessment, Posture Assessment and Security Testing, Penetration Testing and Ethical Hacking.

The successful use of these techniques in industrial software development will validate the results of the research and drive future research. [8]

X. REFERENCES

1. McCabe, T. J. and G. G. Schulmeyer. "System Testing Aided by Structured Analysis: A Practical Experience." IEEE Trans. Software Eng. SE-11, 9 (Sept. 1985), 917-921.
2. Petschenik, N. H. "Practical Priorities in System Testing." IEEE Software 2, 5 (Sept. 1985), 18-23.
3. Powell, P. B. "Planning for Software Validation Verification, and Testing." In Software Validation,
4. Beizer, B. Software System Testing and Quality Assurance. New York: Van Nostrand, 1984.
5. Software Testing by Cognizant Technology Solution.
6. Howden, W. E. Functional Program Testing and Analysis. New York: McGraw-Hill, 1987.
7. Introduction to Software Verification and Validation SEI Curriculum Module SEI-CM-13-1.1 December 1988
8. Paper by Lu Luo available at http://www.cs.cmu.edu/~luluo/Courses/17939Report.pdF
9. Rombach, H. D. Software Specification: A Framework. Curriculum Module SEI-CM-11-1.0, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., Oct. 1987.
10. Budgen, D. Introduction to Software Design. Curriculum Module SEI-CM-2-2.0, Software Engineering Institute, Carnegie Mellon University, burgh, Pa., Nov. 1988.
11. Beizer, B. Software Testing Techniques. New York: Van Nostrand, 1983.
12. Brown, B. J. Assurance of Software Quality. Curriculum Module SEI-CM-7-1.1, Software Engineering Institute, Carnegie Mellon University, Pittsburgh,Pa., July 1987.
13. Petschenik, N. H. "Practical Priorities in System Testing." IEEE Software 2, 5 (Sept. 1985), 18-23
14. Morell, L. J. Unit Testing and Analysis. Curriculum Module SEI-CM-9-1.1, Software Engineering Insti- tute, Carnegie Mellon University, Pittsburgh, Pa., Dec. 1988..
15. Gourlay, J. S. "A Mathematical Framework for the Investigation of Testing." IEEE Trans. Software Eng. SE-9, 6 (Nov. 1983), 686-709.
16. Powell, P. B. "Software Validation, Verification and Testing Technique and Tool Reference Guide." In Software Validation, Verification, Testing, and Documentation, S. J. Andriole, ed. Princeton, N. J.:Petrocelli, 1986, 189-310.