FIPS PUB 101

FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION

1983 JUNE 6

U.S. DEPARTMENT OF COMMERCE/National Bureau of Standards

GUIDELINE FOR LIFECYCLE VALIDATION, VERIFICATION, AND TESTING OF COMPUTER SOFTWARE

ORY: SOFTWARE

ATEGORY: VALIDATION, VERIFICATION, AND TESTING

U.S. DEPARTMENT OF COMMERCE, Malcolm Baldrige, *Secretary*
NATIONAL BUREAU OF STANDARDS, Ernest Ambler, *Director*

Foreword

The Federal Information Processing Standards Publication Series of the National Bureau of Standards (NBS) is the official publication relating to standards and guidelines adopted and promulgated under the provisions of Public Law 89-306 (Brooks Act) and under Part 6 of Title 15, Code of Federal Regulations. These legislative and executive mandates have given the Secretary of Commerce important responsibilities for improving the utilization and management of computers and automatic data processing in the Federal Government. To carry out the Secretary's responsibilities, NBS, through its Institute for Computer Sciences and Technology, provides leadership, technical guidance, and coordination of Government efforts in the development of guidelines and standards in these areas.

Comments concerning Federal Information Processing Standards Publications are welcomed and should be addressed to the Director, Institute for Computer Sciences and Technology, National Bureau of Standards, Washington, DC 20234.

James H. Burrows, *Director*
Institute for Computer Sciences and Technology

Abstract

This Guideline is intended for those who direct or implement software development projects. It recommends that validation, verification, and testing (VV&T) be performed throughout the software development lifecycle, and presents information on selection and use of such techniques to meet project requirements. The Guideline also explains how to develop a VV&T plan to fulfill a specific project's VV&T requirements.

Key words: automated software tools; computer software; Federal Information Processing Standards Publication; software lifecycle; software testing; software validation; software verification; test coverage; test data generation.

Natl. Bur. Stand. (U.S.) Fed. Info. Process. Stand. Publ. (FIPS PUB) 101, 37 pages
(1984)
CODEN:FIPPAT

For sale by the National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161

Federal Information Processing Standards Publication 101

1983 June 6

ANNOUNCING THE

GUIDELINE FOR LIFECYCLE VALIDATION, VERIFICATION, AND TESTING OF COMPUTER SOFTWARE

Federal Information Processing Standards Publications are issued by the National Bureau of Standards pursuant to the Federal Property and Administrative Services Act of 1949, as amended, Public Law 89-306 (79 Stat. 1127), and as implemented by Executive Order 11717 (38 FR 12315, dated May 11, 1973), and Part 6 of Title 15 Code of Federal Regulations (CFR).

Name of Guideline: Guideline for Lifecycle Validation, Verification, and Testing of Computer Software (FIPS PUB 101).

Category of Guideline: Software; Validation, Verification, and Testing.

Explanation: This Guideline presents an integrated approach to validation, verification, and testing (VV&T) that should be used throughout the software lifecycle. Also included is a glossary of technical terms and a list of supporting ICST publications. An appendix provides an outline for formulating a VV&T plan, including the identification of VV&T requirements and the selection of supportive techniques and tools. This Guideline is intended for use by software developers, managers, verifiers, maintainers, and end-users.

Approving Authority: U.S. Department of Commerce, National Bureau of Standards (Institute for Computer Sciences and Technology).

Maintenance Agency: U.S. Department of Commerce, National Bureau of Standards (Institute for Computer Sciences and Technology).

Cross Index: None.

Applicability: This Guideline is intended as a basic reference guide for Federal ADP managers and software developers for ensuring quality software by using validation, verification, and testing procedures during development and operation. Its use is encouraged but is not mandatory.

Implementation: This Guideline should be consulted whenever Federal departments or agencies develop new applications software or undertake major revisions of existing software.

Specifications: Federal Information Processing Standards Publication 101 (FIPS PUB 101), Guideline for Lifecycle Validation, Verification, and Testing of Computer Software (affixed).

Qualifications: This Guideline is planned for use by Federal agencies when they develop new software or undertake major revisions of existing software. The general lifecycle VV&T approach should be implemented but may be augmented or diminished according to project goals and constraints.

Where to Obtain Copies of the Guideline: Copies of this publication are for sale by the National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161. When ordering, refer to Federal Information Processing Standards Publication 101 (FIPS-PUB-101), and title. When microfiche is desired, this should be specified. Payment may be made by check, money order, or NTIS deposit account.

Federal Information
Processing Standards Publication 101

1983 June 6

Specifications for

GUIDELINE FOR LIFECYCLE VALIDATION, VERIFICATION, AND TESTING OF COMPUTER SOFTWARE

Contents

Figures

Tables

1. OVERVIEW

This Guideline presents a methodology of lifecycle validation, verification, and testing (VV&T) for computer software. It is addressed to people associated with software development and maintenance including managers, developers, verifiers, maintainers, and end-users. This Guideline is a basic reference guide for ensuring the production and maintenance of quality software. It recommends that VV&T be performed throughout the software lifecycle.

A software lifecycle is the period of time beginning when the software product is conceived and ending when the resultant software products are no longer available for use. The software lifecycle is typically broken into phases, such as requirements, design, programming and testing, installation, and operations and maintenance. Each phase consists of a well-defined set of activities whose products lead to the evolution of the activities and products of each successive phase. From the outline of the specific lifecycle activities and products of a particular software project, managers can more easily direct, and end-users can examine, the progress of the software development and maintenance. Software developers and maintainers have a well-defined set of tasks to perform. Verifiers, by checking the products of these tasks, can verify that the project requirements are met at each phase.

A VV&T methodology is a procedure of review, analysis, and testing employed throughout the software lifecycle from software planning through the end of software use to ensure the production and maintenance of quality software. Validation determines the correctness of the final program or software with respect to the software requirements. Verification employs integrity and evolution checking to determine internal consistency and completeness. Integrity checking verifies the soundness of the products at each phase of development by analyzing each product for internal consistency and completeness. Evolution checking ensures the completeness and consistency of products at different development phases, where one product is a refinement or elaboration of the other. Testing, either automated or manual, examines program behavior by executing the program on sample data sets.

The term VV&T defines a method incorporating all three techniques for application throughout the software lifecycle to determine functionality, to discover errors, and to ensure the production and maintenance of quality software. Disciplined use of VV&T techniques should permeate all of the development and maintenance processes. A VV&T methodology should also include the review, analysis, and evaluation of intermediate and final products (documents as well as codes) of the lifecycle.

For purposes of illustration, the lifecycle phases used in this Guideline are requirements definition, design, programming and testing, installation, and operations and maintenance. Section 2 presents generic VV&T activities that should accompany each of these phases. Descriptions of development and maintenance activities are also included in the text so that VV&T is placed in its appropriate perspective. Specific techniques for implementing a VV&T approach are dependent upon a project and its development method; hence, specific techniques vary with a project. However, the VV&T activities summarized in figure 2.1 should occur for all projects. The integration of a VV&T methodology with the overall project, beginning at the requirements phase, is essential in producing and maintaining quality software.

No single VV&T technique can guarantee correct, error-free software. However, a carefully chosen set of techniques for a specific project can help to ensure the development and maintenance of quality software for that project. Section 3 provides guidance in selecting and combining different types of techniques to form an effective VV&T program. Static, dynamic, and formal analyses are discussed and guidance for their use provided. Figures establishing three different levels of recommended VV&T approaches are also included.

A VV&T program should be tailored to the needs and constraints associated with the software project. An outline for developing a VV&T program is presented in the appendix. It indicates the information that should be included and can also be used as a checklist to determine if appropriate planning is being done and to ensure that necessary decisions are recorded.

Further aids for the understanding of VV&T concepts, lists of techniques and tools, and details on VV&T planning are available from the supporting documents listed in this Guideline. A glossary provides definitions for some of the more frequently used VV&T terms.

2. LIFECYCLE VALIDATION, VERIFICATION, AND TESTING

VV&T is a process of review, analysis, and testing employed throughout the software lifecycle to ensure the production of quality software. The review and analysis should include the examination of the development product and the documentation at each phase. Figure 2.1 presents an overview of the VV&T activities that should accompany each phase of development. This summary provides a framework from which a VV&T program can be tailored for specific projects. Each lifecycle phase is comprised of both development and VV&T activities. In order to emphasize their relationships to each other, the following sections elaborate on both the development and VV&T lifecycle activities and their products. Uppercase titles are used for VV&T activities and products for which the VV&T team is responsible. The VV&T team may be members of the development group, the same organization, or an independent group.

LIFECYCLE VV&T ACTIVITIES
I. Requirements Definition and Analysis Phase
* Development of the project VV&T plan
* Generation of requirements-based test cases
* Review and analysis of the requirements
* Review and analysis of the draft user manual
II. Design Phase
* Completion of VV&T plan
* Generation of design-based test scenarios
* Review and analysis of the design
Preliminary design integrity check
Preliminary design evolution check
* Development of test support software
III. Programming and Testing Phase
* Completion of test case specification
* Review, analysis, and testing of the program
Code integrity check
Code evolution check
Unit test
Integration test
System test
IV. Installation Phase
* System acceptance
V. Operations and Maintenance Phase
* Software evaluation
* Software modification evaluation
* Regression testing

Figure 2.1 *Summary of VV&T activities*

2.1 Requirements Definition and Analysis Phase

DESCRIPTION: The goal of the requirements phase is to specify both the problem and the constraints upon the solution in a rigorous form. Requirements identification is somewhat iterative with the requirement statement being subject to modification during design as the problem is better understood.

These modifications must be documented to create a traceable record of the progress and evolution of the final product. Two planning activities occur during this phase: (1) project plans, budgets, and schedules are developed; and (2) a VV&T plan is developed from the VV&T requirements identified in this phase.

DEVELOPMENT PRODUCTS:

* The Software Requirements Document: This document specifies what the system must do, including the requisite information flows, processing functions, performance constraints, and the acceptance criteria for deciding that specified requirements are satisfied. This document also contains those internal specifications which, although transparent to the end user, are necessary to the development of the end product. (Development Product)

* The Project Plan: The project plan explains the strategy for managing the development of the software. This document defines the goals and activities for all phases of the project, estimates resource requirements, and specifies intermediate milestones, including management and technical reviews. It defines methods for design, coding, VV&T, documentation, problem reporting, and change control. In particular, it assigns responsibility for the VV&T effort, depending on project size, criticality, and budget. The responsible party may be the programmer, a separate member of a development group, member(s) outside the development group but from the same organization, or from a completely independent organization. The project plan also specifies supporting techniques and tools. (Development Product)

* Project Standards: Project standards define specific techniques and formats for requirements, design, coding, languages, documentation, configuration management, and VV&T. (Development Product)

* Draft of Users' Manual: A users' manual describes in non-ADP terminology how to use the system. The manual describes both the system functionality and the user interface. Its preparation during the requirements phase is an excellent mechanism for ensuring that both the users and the developers share the same view of the system. The manual serves as a reference document for the preparation of input data and parameters and for interpretation of results. (Development Product)

VV&T ACTIVITIES AND PRODUCTS:

* DEVELOPMENT OF THE PROJECT VV&T PLAN: During this activity, the VV&T analyst (who may be part of the development group or from a separate organization) will determine VV&T requirements; design a VV&T process; select techniques and tools; and establish schedules, responsibilities, and budgets. (VV&T Activity)

* THE VALIDATION, VERIFICATION, AND TESTING PLAN: The VV&T plan specifies goals and approaches to the VV&T activities. It contains the outline for a project specific VV&T process, identifies techniques and tools to be used, and specifies plans (schedules, budgets, responsibilities, etc.) for performing the VV&T activities. (VV&T Product)

* INITIAL SOFTWARE TEST CASE SPECIFICATION: A basic set of test cases is developed to clarify and to determine measurability of each software requirement. The acceptance criteria are used to develop the test cases. Input data and expected results for each test case are included in the specification. (VV&T Activity, Product)

* REVIEW AND ANALYSIS OF THE PROJECT REQUIREMENTS: Project requirements are reviewed for clarity, completeness, consistency, testability, and traceability to the problem statement. The goal of this activity is to ensure that these requirements will result in a practical, usable solution to the entire problem. (VV&T Activity)

* REVIEW AND ANALYSIS OF USERS' MANUAL: The users' manual is reviewed for clarity and consistency. It is checked for completeness against the requirements document. In addition, this verification activity includes ensuring that the internal specifications of the requirements document are defined sufficiently to lead to the production of the functions and interfaces described in the users' manual. (VV&T Activity)

2.2 Design Phase

DESCRIPTION: The goal of this phase is to design a solution that satisfies the requirements and constraints. Alternative solutions are formulated and analyzed and the best solution is selected and refined. A high-level specification which defines information aggregates, information flows, and logical processing steps is generated and is refined into a detailed specification describing the physical solution (algorithms and data structures). The result is a solution specification that can be implemented in code with little additional refinement. Project plans (schedules, budgets, deliverables, etc.) are reviewed and revised as appropriate.

DEVELOPMENT PRODUCTS:

* The Design Specification: Frequently this specification contains two documents: (1) a preliminary design document to identify a high-level solution developed during this phase and (2) a detailed design document which defines and refines software (algorithms and data) to be coded in the following phase. (Development Product)

* A Revised Requirements Specification: Design activities may reveal incorrect, inconsistent, infeasible, or ambiguous requirements resulting in the revision of their specification. (Development Product)

* An Updated Project Plan: Upon completion of the preliminary design, the scope and complexity of the solution should be well understood. As a result, the project plan (schedules, budgets, deliverables, etc.) is more accurate and realistic. (Development Product)

VV&T ACTIVITIES AND PRODUCTS:

* AN UPDATED VV&T PLAN: New or revised project requirements may warrant revision of the VV&T plan. The detailed design plan may indicate the need for additional testing procedures. (VV&T Product)

* REVIEW AND ANALYSIS OF THE DESIGN: The design is analyzed to ensure internal consistency, completeness, correctness and clarity, and to verify that the design, when implemented, will satisfy the requirements. (VV&T Activity)

* SOFTWARE TEST CASE SPECIFICATION: Additional test scenarios and test cases (input data and expected results) are developed to exercise and test logical and structural aspects of the design. (VV&T Product)

* IMPLEMENT OR ACQUIRE TESTING SUPPORT TOOLS: Development or acquisition of any support software needed for unit, integration, or system testing should be completed and installed during the detailed design phase to ensure readiness during programming and testing. (VV&T Activity)

2.3 Programming and Testing Phase

DESCRIPTION: During this phase, the detailed design is implemented in code, resulting in a program or system ready for installation. Three types of testing are performed: unit, integration, and system. Although the programmer is responsible for unit testing, the responsibility for integration and system testing is determined by the project management, depending on project size and criticality. The project plan contains general information, and the VV&T plan specific details, assigning responsibilities for the development, execution, and evaluation of all test cases and data at the various levels of testing. For large or critical software, separate test teams may be used. Unit testing checks for typographic, syntactic, and logical errors. Code modules are checked individually by the programmers who wrote them to ensure that each correctly implements its design and satisfies the specified requirements. Integration testing focuses on checking the intermodule communication links and on testing aggregate functions formed by groups of modules. System testing examines the operation of the system as an entity, sometimes in a simulated operating environment. This type of testing ensures that the software requirements have been satisfied both singly and in combination. The final activity of this phase is to ensure readiness for the software installation, including revision of plans as necessary and completion of all other coding, testing, and documentation.

DEVELOPMENT PRODUCTS:

* Program Code: Fully documented and tested code is constructed, ready for installation. (Development Activity, Product)

* User Documentation: Manuals describing the input and report formats, user commands, error messages, and instructions for operation by the user are completed. (Development Product)

* Maintenance Manual: Documentation to maintain the system is written; however, the manual may be modified or completed during the installation phase. (Development Product)

* Installation Plan: Such a plan specifies the approach to, and details of, the installation of the software. (Development Product)

* Problem Reports: Observed problems are recorded in formal statements and may require return to a previous phase for resolution. (Development Product)

VV&T ACTIVITIES AND PRODUCTS:

* SOFTWARE TEST CASE SPECIFICATION: Final revisions and additions to the test data are made. (VV&T Activity, Product)

* REVIEW AND ANALYSIS OF THE PROGRAM: This activity includes checking for adherence to coding standards and manual/automated analysis of the program by static, dynamic, and formal methods. (VV&T Activity)

* TESTING THE PROGRAM: The program is executed with the test data; actual results are compared with the expected results and are validated for satisfaction of the requirements. (VV&T Activity)

* TEST RESULTS AND TEST EVALUATION REPORTS: The testing activities, including comparison of actual and expected results, are documented. (VV&T Product)

* PROBLEM REPORTS: Observed problems are recorded in formal statements and may necessitate returning to a previous phase for resolution. (VV&T Product)

2.4 Installation Phase

DESCRIPTION: During this phase the system is placed into operation. The first task, integrating the system components, may include installing hardware, installing the program(s) on the computer, reformatting/creating the data base(s), and verifying that all components have been included. Modification of the program code may be necessary to obtain compatibility between hardware and software, or between different software modules for which earlier simulation testing may not have been adequate. The next task is to test the system in its complete operating environment. The test data from earlier phases is enhanced and used. The result is a system qualified and accepted for production use. The third task is the start of system operation. If a previous system exists, then strategies for its replacement include immediate total replacement, "phasing-in" of the new system, or parallel operation of both systems. A completely new program could either be phased into operation or could be implemented at once. This task also includes operator and user training.

DEVELOPMENT PRODUCT:

* Installation Report: This report describes the results of the installation activities, including data conversion, installation testing/results, and software/system problems and modifications. (Development Product)

VV&T ACTIVITIES AND PRODUCTS:

* ACCEPTANCE TESTING: Once the system is tested, the primary VV&T activity centers on acceptance of the system by the customer (or principal user when the developers and users are the same). Acceptance may range from review or acknowledgment of the VV&T activities during system development to detailed acceptance testing by the customer prior to formal acceptance. (VV&T Activity)

FORMAL ACCEPTANCE: A customer representative should formally sign off on a form indicating that testing has been completed and that the system is accepted. (VV&T Product)

2.5 Operations and Maintenance Phase

DESCRIPTION: This phase involves the actual use of the software and monitoring of its operation to ensure that it succeeds in solving the user's problem. Most often, some need for modifying the software arises during this phase. The maintenance process involves determining the cause for each modification which could be an error made in the original development or previous maintenance, a change in the surrounding environment, the recognition of a new or evolving requirement, or the desire for a design modification to improve performance, usability, etc. Once the cause is determined, the software (code and documentation) is "redeveloped" from that point. For example, redevelopment due to a change in requirements would result in modifications to the requirements specification, the design, the code, and user and operation manuals. Problem reporting, change requests, and other change control mechanisms are used to facilitate the systematic correction and evolution of the software. In addition, performance measurement and evaluation activities are performed to ensure that the system continues to meet performance requirements in the context of a changing system environment.

DEVELOPMENT PRODUCTS:

* Problem Reports: These are formal statements of observed problems. Their analyses may result in software change requests. (Development Product)

* Change Requests: These are formal requests for specific modifications to the software. These could be generated due to an error (i.e., problem report) or a modification of the requirements or design. (Development Product)

* Revision to Initial Development Products: As a result of change requests, any one or all of the products of the initiation and development phases may require revision. (Development Product)

VV&T ACTIVITIES AND PRODUCTS:

* SOFTWARE EVALUATION: Continuous monitoring and evaluation to assess the operation of the software and to ensure continued satisfaction of user requirements occurs throughout the operation and maintenance of the software. (VV&T Activity)

* CHANGE REQUESTS: Formal requests by VV&T personnel for specific changes to the software must be submitted to those responsible for making the revisions. (VV&T Product)

* REGRESSION TESTING: Test cases which a program has previously executed correctly in order to detect errors created during software modification are rerun and compared. (VV&T Activity)

* SOFTWARE MODIFICATION EVALUATION: Requested modifications to the system are evaluated in the same manner that the original software development was evaluated. If the requirements or design specifications are modified, the VV&T activities appropriate to those phases should be performed. When the modifications are completed, they must be reviewed and tested to ensure that they not only fulfill the modification request, but also have not adversely affected any other part of the system. (VV&T Activity)

3. SELECTION AND COMBINATION OF TECHNIQUES

Software VV&T detects errors and validates that the product is correct, complete, and consistent with respect to its requirements. However, no single VV&T technique can guarantee correct, error-free software. A combination of carefully-applied techniques can provide confidence in the adequacy of the software. Three types of analysis (static, dynamic, formal) are available and each provides the VV&T analyst with different types of specific information about the solution being examined.

1. Static analysis detects errors through the examination of the product. It focuses on the form and structure of the solution, but not the functional or computational aspects. It is also the technique used to examine all document items at all phases of development.

2. Dynamic analysis is the process of determining the validity of a program and of detecting errors by studying the program's response to a set of input data. It addresses the functional, structural, and computational aspects.

3. Formal analysis uses rigorous mathematical techniques to analyze the algorithms or properties of a solution. It can provide a strong statement regarding certain properties of a solution including correctness, but is limited by the difficulty of application and lack of automated support.

These three types of analysis should be used in conjunction with one another to provide a powerful VV&T technology.

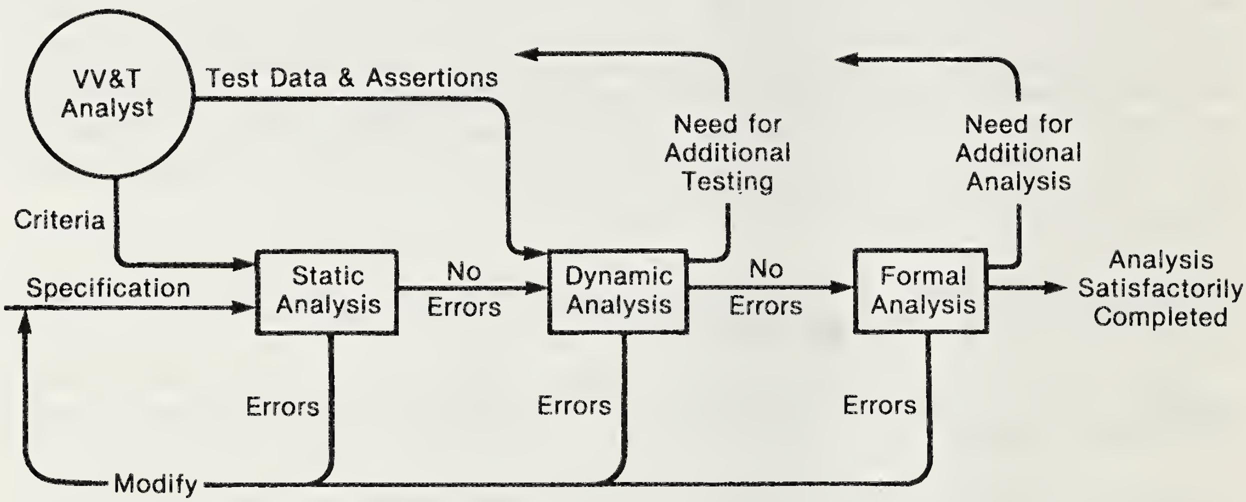

Figure 3.1 *General VV&T integration strategy*

The integration strategy shown in figure 3.1 is simple. First, static analysis techniques are applied to analyze the form of the specification. These techniques are straightforward and usually the least expensive to apply. Applicable to all levels of specification, they identify flaws that could prevent application of dynamic and formal techniques. However, dynamic analysis methods are needed to focus on the functional meaning of the solution and to detect errors in their specification. These may be manually applied to the requirements and design specifications. The code may undergo dynamic testing by executing test data on it. Dynamic analysis techniques, when applied properly, are effective, comprehensive, and within the resource constraints of nearly all projects. For further assurances, formal analysis techniques may be used; these are usually quite expensive because they require highly trained people and sophisticated support.

The analysis techniques discussed above apply to different phases of the lifecycle. In figures 3.5 through 3.7, three levels of recommended combinations of VV&T techniques are presented. A VV&T approach appropriate to the software requirements and resources of the project should be used. All recommendations are cumulative. For example, the comprehensive set of techniques includes the basic set. NBS Special Publications 500-75 and 500-93 contain information on specific techniques and tools that can be used to support lifecycle VV&T.

3.1 Requirements Definition and Analysis

During the requirements phase, static analysis focuses on checking adherence to specification conventions, consistency, completeness, and language syntax. Dynamic analysis focuses upon information flows, functional interrelationships, and performance requirements. Manual methods such as inspections, peer reviews, and walkthroughs are effective in accomplishing both types of analysis if rigorously performed. If the constructs of the requirements specification scheme are clearly defined and capable of being represented in a computer processable form, then automated tools may be used to perform both the static and dynamic analyses. Several such specification methods with supporting tools are available.

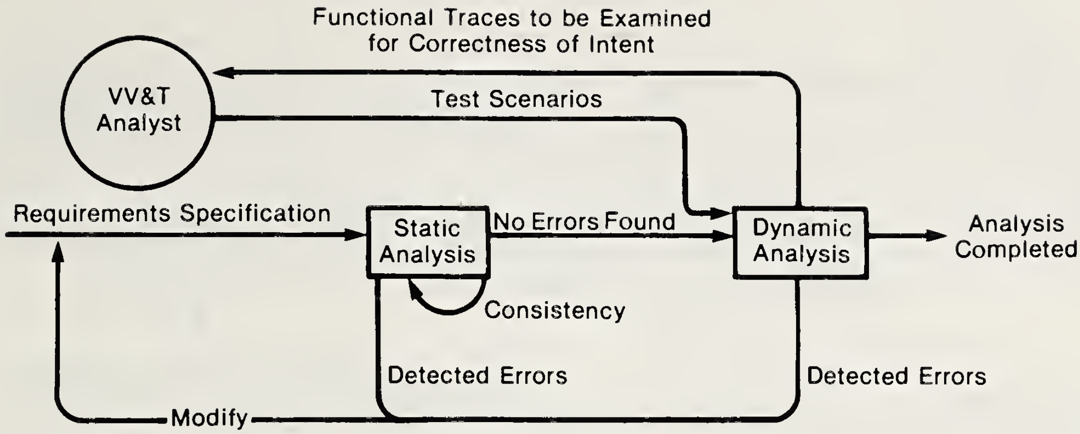

Figure 3.2 *Integrated approach to requirements VV&T*

3.2 Design

As with the requirements, the representation schemes used to specify the design determine the specific analysis techniques which should be employed. Design specification schemes generally provide mechanisms for specifying algorithms and their inputs and outputs in terms of modules. Inconsistencies in specifying the flow of data objects through the modules can be detected by static analysis techniques. Certain errors made during the composition of a design can be detected, such as the inconsistencies between the inputs and outputs specified for a high level module and the cumulative inputs and outputs of the submodules.

Dynamic analysis of a design is accomplished by some form of design simulation. This may be a manual walkthrough or an automated simulation using a model of the design. Manual walkthroughs, when rigorously performed and guided by documented test scenarios, are an effective technique for analyzing a software design. For larger software designs and highly critical systems or components, an automated simulation may be appropriate. This requires the construction and execution of a solution model with the test scenarios. To be credible the model must be validated as a faithful representation of the solution, although the higher the required degree of model fidelity, the higher the cost of simulation. This cost generally increases with the complexity of the model.

Formal analysis techniques may be manually applied to a design specification if the specification is sufficiently formal and exact. This involves tracing paths through the design specification and formulating a composite function for each. This procedure is more feasible at higher levels of a hierarchical design specification. Less detail is present and the resulting algorithm paths are relatively short and few in number. Thus, the evolved functions remain concise and manageable. The purpose of deriving these composite functions for a given level of design is to compare them to the functions of the previous level. This process ensures that the design continues to specify the same functional solution as is hierarchically elaborated.

The formal analysis of a design specification can be improved by using automated symbolic execution tools. Such tools can be expensive to create and operate; in return, however, they offer greater speed and capacity for manipulating detailed specifications. Thus, the functional effects of all levels of a design specification can be determined.

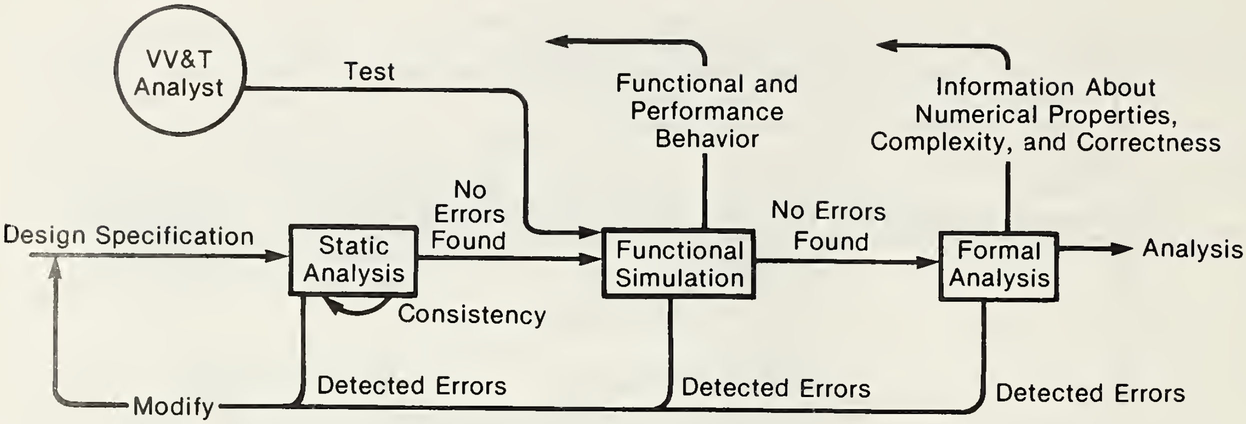

Figure 3.3 *Integrated approach to design VV&T*

3.3 Programming and Testing

Static analysis techniques and tools are used to ensure the proper form of programming products, for example, code and documentation. This can be accomplished by checking adherence to coding and documentation conventions, interface and type checking, etc. The checking can be done by manual techniques and automated tools. Inspections and code auditors fit into these categories, respectively. Dynamic analysis techniques are employed to study the functional and computational correctness of the code. Initially, such manual techniques as walkthroughs can be used as an effective forerunner to testing. Testing is accomplished by running the code on the test data sets which were developed during the requirements and design phases and completed during the programming and testing phase. The correctness of the test executions is determined more definitively when the expected results are specified. Testing for adherence to assertions is also highly advisable. These assertions, are products of the design activity and provide additional information regarding expected behavior of the software.

If software is being developed in an environment other than the production environment, testing is more problematic. Here the production environment can be simulated or taken into account informally. In any case, the validity of the test results depends upon the fidelity of the simulation or informal judgments. If there is a significant difference in the two environments, there will be an eventual need for some additional testing in the actual production environment. The balance between simulation testing and actual production environment testing must be determined for each individual project, based partially upon how available and expensive the production environment is.

Whenever assurances of correctness over and above those provided by dynamic analysis are required, formal analysis follows testing. Symbolic evaluation and formal proof techniques can be effective in achieving high levels of confidence. An integrated VV&T approach is shown in figure 3.4.

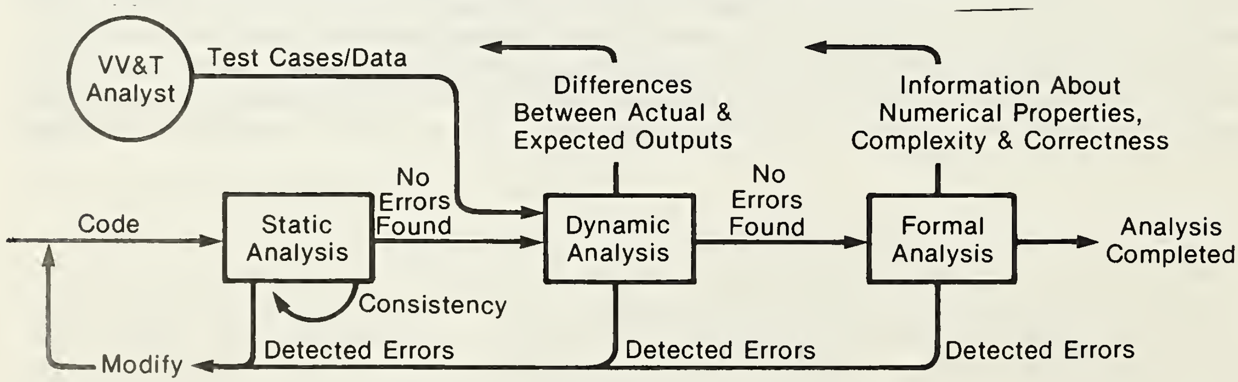

Figure 3.4 *Integrated approach to code VV&T*

3.4 Installation

During the installation phase, testing is done to verify earlier test results, to test special cases, and to determine whether or not to accept the system. In the first case, samples of earlier tests from any phase and technique are selected and rerun. This gives added assurance that the tests were accurate when first used and that their results were not negated at a later stage of development. If earlier testing required simulation, then some special tests may be run to verify those results in the actual production environment. Situations unique to the operating environment are examined at this stage. Formal acceptance testing is performed, to the extent required by the project. Such testing may include functional tests and trial use of the user documentation or training.

3.5 Operations and Maintenance

During operations and maintenance, any problems within the system, additions and enhancements to it, or modifications due to environmental changes involve the use of techniques appropriate to the development phases that are affected.

3.6 Recommended Techniques

The methodology of validation, verification, and testing (VV&T) throughout the lifecycle of computer software requires the integration of development activities with VV&T activities. The VV&T requirements are tailored specifically to a project, and its requirements, constraints, and resources. Methodologies may range from simple for small projects to very complex for large, and/or highly critical projects. Disciplined application of a VV&T methodology developed from careful selection and combination of VV&T techniques can help ensure the production of high quality software. All recommendations are cumulative; for example, the comprehensive set of techniques assume the inclusion of the basic set.

Phase	Technique
Requirements	Review
Design	Inspection
Code	Inspection Test Coverage Unit: 100% statement Integration: 100% module call System: 95% module call 100% of major logic paths
Installation	Acceptance Testing: Insure continued validity of system test
Operations and maintenance	For affected code: Inspection Test Coverage: 100% statement 100% module

Figure 3.5 *Recommended techniques for lifecycle VV&T (basic approach)*

Phase	Technique
Requirements	Inspection
Design	Interface Analysis Data Flow Analysis
Code	Assertions Standards Audit Interface Analysis Data Flow Analysis Explicit Trace-back of Code to Requirements
Installation	Acceptance Testing
Operations and maintenance	For affected code: Reapply techniques used during development

Figure 3.6 *Recommended techniques for VV&T (comprehensive approach)*

Phase	Technique
Requirements	Automated Consistency Analysis
Design	Automated Consistency Analysis Automated Simulation Proof of Critical Sections
Code	Symbolic Evaluation Proof of Critical Sections or Properties
Installation	Acceptance Testing: System Certification
Operations and maintenance	Re-do proofs that cover affected areas; retest

Figure 3.7 *Recommended techniques for VV&T for critical software*

Note: All recommendations are cumulative, for example, the comprehensive set of techniques assume the inclusion of the basic set.

SUPPORTING ICST DOCUMENTS

* NBS Special Publication 500-56 "Validation, Verification, and Testing for the Individual Programmer," M. Branstad, J. Cherniavsky, and W. Adrion, 1980.

* NBS Special Publication 500-75 "Validation, Verification, and Testing of Computer Software," W. Adrion, M. Branstad, and J. Cherniavsky, 1981.

* NBS Special Publication 500-87 "Management Guide to Software Documentation," A. Neumann, 1982.

* NBS Special Publication 500-88 "Software Development Tools," R. Houghton, Jr., 1982.

* NBS Special Publication 500-93 "Software Validation, Verification, and Testing Technique and Tool Reference Guide," P. Powell, Editor, 1982.

* NBS Special Publication 500-98 "Planning for Software Validation, Verification, and Testing," P. Powell, Editor, 1982.

** FIPS 38 "Guidelines for Documentation of Computer Programs and Automated Data Systems," 1976.

** FIPS 64 "Guidelines for Documentation of Computer Programs and Automated Data Systems for the Initiation Phase," 1979.

NOTES:

1. Subsequent NBS documents will include guidance on acceptance testing and maintenance.
2. NBS documents may be ordered from:

* Superintendent of Documents
U.S. Government Printing Office
Washington, DC 20402
(202) 783-3238

** National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161
(703) 487-4650

REFERENCES

[CAIN] Caine, S. H.; Gordon, E. K. PDL: A tool for software design. Proceedings of the National Computer Conference; 1975.

[SAMM] SAMM (Systematic Activity Modeling Method) Primer. BCS 10167; 1978 October.

[TEIC] Teichroew, D.; Hershey, E. PSA/PSL: A computer-aided technique for structured documentation of information processing systems. IEEE Transactions on Software Engineering, Vol. SE-3: No. 1; 1977.

GLOSSARY

ACCEPTANCE TESTING: formal testing conducted to determine whether a software system satisfies its acceptance criteria and to enable the customer to determine whether to accept the system.

ASSERTION: a logical expression specifying a program state that must exist or a set of conditions that program variables must satisfy at a particular point during program execution.

CERTIFICATION: acceptance of software by an authorized agent usually after the software has been validated by the agent, or after its validity has been demonstrated to the agent.

COMPLETENESS: the property that all necessary parts of the entity in question are included. Completeness of a product is often used to express the fact that all requirements have been met by the product.

CONSISTENCY: the property of logical coherency among constituent parts. Consistency may also be expressed as adherence to a given set of rules.

CORRECTNESS: the extent to which software is free from design and coding defects, i.e., fault free. It is also the extent to which software meets its specified requirements and user objectives.

DATA FLOW ANALYSIS: a graphical analysis technique to trace behavior of program variables as they are initialized, modified, or referenced while the program executes.

DEBUGGING: the process of correcting syntactic and logical errors detected during coding. With the primary goal of obtaining an executing piece of code, debugging shares certain techniques and strategies with testing but differs in its usual ad hoc application and local scope.

DYNAMIC ANALYSIS: involves execution or simulation of a development phase product. It detects errors by analyzing the response of a product to sets of input data.

EVOLUTION CHECKING: testing to ensure the completeness and consistency of a software product at different levels of specification, where one product is a refinement or elaboration of another.

FORMAL ANALYSIS: use of rigorous mathematical techniques to analyze the algorithms of a solution. The algorithms may be analyzed for numerical properties, efficiency, and/or correctness.

FUNCTIONAL TESTING: application of test data derived from the specified functional requirements without regard to the final program structure.

INSPECTION: a manual analysis technique which examines the program (requirements, design, or code) in a very formal and disciplined manner to discover errors.

INTEGRATION TESTING: orderly progression of testing in which software elements, hardware elements, or both, are combined and tested, until all intermodule communication links have been integrated

INTEGRITY CHECKING: testing to verify the soundness of a software product at each phase of development.

INTERFACE ANALYSIS: checking that intermodule communication links are performed correctly.

LIFECYCLE: see SOFTWARE LIFECYCLE

PROOF OF CORRECTNESS: use of techniques of mathematical logic to infer that a relation between program variables assumed true at program entry implies that another relation between program variables holds at program exit.

REGRESSION TESTING: Rerunning test cases which a program has previously executed correctly to detect errors created during software correction or modification activities.

SIMULATION: use of an executable model to represent the behavior of an object. During testing, the computational hardware, the external environment, and even code segments may be simulated.

SOFTWARE: computer programs, procedures, rules, and possibly associated documentation and data pertaining to the operation of a computer system.

SOFTWARE LIFECYCLE: period of time beginning when a software product is conceived and ending when the product is no longer available for use. The software lifecycle is typically broken into phases, such as, requirements, design, programming and testing, installation, and operation and maintenance.

STANDARDS AUDIT: check to ensure that applicable standards are used properly.

STATEMENT TESTING: a test method satisfying the criterion that each statement in a program be executed at least once during program testing.

STATIC ANALYSIS: direct analysis of the form and structure of a product without executing the product. It may be applied to the requirements, design, or code.

SYMBOLIC EXECUTION or EVALUATION: an analysis technique deriving a symbolic expression for each program path.

SYSTEM TEST: process of testing an integrated hardware and software system to verify that the system meets its specified requirements.

TESTING: examining the behavior of a program by executing the program on sample data sets.

UNIT TEST: testing of a module for typographic, syntactic, and logical errors, for correct implementation of its design, and for satisfaction of its requirements.

VALIDATION: determination of the correctness of the final program or software produced from a development project with respect to the user needs and requirements.

VERIFICATION: the demonstration of consistency, completeness, and correctness of the software at each stage and between each stage of the development lifecycle.

VV&T: validation, verification, and testing; used as an entity to define a procedure of review, analysis, and testing throughout the software lifecycle to discover errors, determine functionality, and ensure the production of quality software.

WALKTHROUGH: a manual analysis technique in which the module author describes the module's structure and logic to an audience of colleagues.

NOTE: Most of the definitions above and many more terms common to VV&T and software practices appear in:

ADRION, W. R.; BRANSTAD, M. A.; CHERNIAVSKY, J. C. Validation, verification, and testing of computer software. Natl. Bur. Stand. (U.S.) Spec. Publ. 500-75.

IEEE Computer Society, Technical Committee on Software Engineering. Glossary of software engineering terminology (Draft-IEEE Project 729), The Institute of Electrical and Electronics Engineers, Inc., 345 East 47th St., New York, NY 10017.

APPENDIX A
PLANNING FOR VALIDATION, VERIFICATION, AND TESTING

A validation, verification, and testing plan is a document, or group of documents, specifying a project's VV&T requirements and the procedures needed to achieve them. An outline of the plan may be general and brief, or detailed as shown in figure A.1. Because the general planning drives the VV&T planning, in turn providing feedback to the overall development, the general project planning and the VV&T planning are closely integrated. Once the general background, goals, and requirements are clearly understood, the VV&T planning begins. The following four-step approach is useful in developing a project's VV&T plan:

(1) identify the VV&T requirements
(2) determine the constraints on the VV&T activities
(3) select VV&T techniques
(4) itemize results of the first three steps in a written VV&T document.

Some factors to consider during VV&T planning are the following:

* VV&T requirements are based on project needs and constraints.
* The VV&T techniques and tools that can be used are dependent upon and must be consistent with the project's development approach.
* The details of the VV&T plan, e.g., time and resource requirements, must be coordinated with the overall schedule and budgets.
* Planning activities take place during the requirements phase, with attention paid to activities that require long lead time or must begin early in the project, such as personnel training or the initiation of tool acquisition.
* Revisions and refinements of the plan may occur during the design phase.
* A small project may have a brief plan; however, as the size, complexity, and critical nature of the project increase, so will the complexity and formality of its VV&T plan and the effort required to develop it.

The outline in figure A.1 indicates the contents of a VV&T plan. The project's background and requirements, as well as the information from the first three steps of the four-step approach are included. Section 1 contains the general project background and information on the proposed solution. Section 2 specifies the VV&T requirements, measurement criteria, and constraints. Section 3 states the VV&T procedures to be applied during development in general and by phase. Supporting information for the selections made in Section 4 is detailed in Appendix B of the plan. Further information on planning a VV&T methodology may be found in NBS Special Publication 500-98, Planning for software validation, verification, and testing.

I. Background and Introduction

Establishes the context for the VV&T document. Is brief and introductory in nature. Focuses on those aspects of the problem and/or solution which influence the VV&T needs and approach.

A. Statement of problem
B. Proposed solution
C. References/related documents

II. VV&T Requirements and Measurement Criteria

Presents the VV&T requirements in one of several formats: the total VV&T requirements, with all worksheets and phase information; a summary of requirements information; statement of project level information, with phase data presented later.

A. VV&T requirements and their importance
  1. Functional
  2. Performance
  3. Reliability
  4. Other
B. Measurement criteria for each requirement
  1. General
  2. Product specific
  3. Phase specific
C. References/related documents

III. Phase by Phase VV&T Plans

First, describes VV&T approach by phases, products, major reviews and checkpoints, and practices common to all phases. Then, presents the specific activities to be carried out phase by phase.

A. Project background and summary information
  1. Project phases and products
  2. Major reviews (both management and technical)
B. Requirements phase VV&T activities
  1. VV&T activities
  2. VV&T techniques and tools selected
    a. Reviews
    b. Methods of analysis
  3. Required support tools, automated & other
  4. Roles and responsibilities
  5. Schedules
  6. Budgets
  7. Personnel
C. Design phase
D. Programming and testing phase
E. Installation phase
F. Operations and maintenance phase

(C–F contain items 1–7, as indicated in B, as needed)

Appendix A Project and Environmental Considerations

A. Technical issues
B. Project constraints
C. Computing resources

Appendix B Technique and Tool Selection Information

A. Candidate list of techniques and tools
B. Rationale for selection of techniques and tools

Figure A.1 *A detailed outline of a project's VV&T plan*

APPENDIX B
EXAMPLE APPLICATIONS OF VALIDATION, VERIFICATION, AND TESTING TECHNOLOGY

This appendix presents examples in which the concepts of software development, software VV&T, and VV&T planning are illustrated. The purpose is to show how these concepts may be applied in a variety of situations.

Two examples are presented, which use an automobile insurance transaction processing procedure as the system being developed. These examples illustrate adaptation of both the basic and the comprehensive VV&T approaches to specific projects. These examples cover only the development phases, with the design phase subdivided into a preliminary design and a detailed design. The VV&T techniques for these examples differ slightly from the recommended VV&T approaches of figures 3.5 and 3.6. These differences illustrate that a VV&T methodology may be tailored to fit the goals and constraints of a specific software project.

B.1 Overview of Examples

Example 2 builds upon Example 1. The tools introduced in Example 2 are to be used in addition to the techniques described in the first example. Figure B.1.1 presents an overview of the different VV&T tools and techniques which are used in the examples.

Software Development

	Example (#1) Basic Techniques	(#2) Comprehensive
Supporting Technology	Graphical Requirements Representation	
Static Analysis	Walkthroughs Reviews Inspection	Interface Checker Dataflow Analyzer Standards Checker
Dynamic Analysis	Functional Testing Test Coverage Analysis	Assertion Generation Assertion Checking
Formal Analysis		

Figure B.1.1 *Overview of examples*

The software development subphases for each example are:

- o Requirements,
- o Preliminary design,
- o Detailed design, and
- o Programming (includes testing).

Each of the examples will be presented showing for each phase:

- o Inputs to the phase,
- o Outputs from the phase,
- o Supporting technology used in the phase, and
- o Activities which comprise the phase.

Most activities will contain:

- o VV&T purpose for the activity,
- o VV&T technique(s) used by the activity, and
- o Example(s).

Tables B.1 and B.2 provide a summary of the development and VV&T techniques and activities for the basic and comprehensive activities. These tables present a synopsis of the examples.

Table B.1 *Example 1. Summary software development using basic VV&T techniques*

Subphases	*Requirements*	*Preliminary design*	*Detailed design*	*Programming*
INPUT	•Informal prose requirements	•Detailed requirements specification –Revised prose description –Revised graphical GR representation •VV&T plan	•Preliminary design document •VV&T plan •Test cases	•Detailed design document •VV&T plan •Test cases
OUTPUT	•Detailed requirements specification •VV&T plan •Initial test cases	•Preliminary design document –Further refined GR system representation –Detailed user input/ output specification –Basic control flow design –Basic system information specification •Additional test cases	•Detailed design document •Additional test cases	•System software •Test results
SUPPORTING TECHNOLOGY	•Formal requirements reviews •A graphical requirements representation method •Requirements-based functional testing	•Reviews •A graphical requirements representation method •Design-based functional testing	•Reviews •Database management system (DBMS) •Design-based functional testing	•Cross reference •Compilers •Database management system •Operating system •Reviews •Test coverage analyzer
ACTIVITIES	•Initial requirements reviews •Requirements analysis •VV&T planning •Initial test case generation •Interaction with customer •Sign-off by customer	•Refinement of graphical representation •Specify information design •Design program architecture & allocate requirements •Design basic control flow •Test case generation •Preliminary design review •Traceback	•Detailed database design •Detailed module design •Test case generation •Design review •Design inspection •Traceback	•Code development •Module testing •Function testing •Test coverage analysis •Traceback

The application area used in the examples is representative of a large number of Government and commercial systems. Transaction processing systems are perhaps the most common of all commercial systems. Many banking, billing, payroll, inventory, and insurance applications are in this category. Thus, the four examples focus on this area.

The transaction processing system is set in the context of an auto insurance application. In order to limit the size of the presentations some simplifications have been made in the application area. An expert in the auto insurance field will surely detect omissions and simplifications in details of the system as described. The reader is encouraged, however, to not focus on the application area, but rather on the VV&T principles applied. The details provided enable presentation of specific instances of the application of VV&T techniques.

The Auto Insurance Management System (AIMS) described in the examples supports all the major activities of such a company: accounts payable (claims processing), accounts receivable (premium processing), management reports, and database management. AIMS must issue client premium due notices, checks to repair shops (or clients), recommend policies that should be cancelled, monitor the company's day-to-day financial health, and so forth. Further details of the system's requirements are included in the first example.

B.2 EXAMPLE 1: Software Development Using Basic VV&T Techniques

In this example the details of the AIMS are presented in addition to the actual manual VV&T practices which are applied within each of the four phases of the software development lifecycle.

B.2.1 Requirements Subphase Activity Descriptions

B.2.1.1 Initial Requirements Review

The informal prose requirements for the AIMS is given in figure B.2.1. Appropriate management and technical personnel from the software development group review these requirements for completeness, consistency, and correctness and prepare a list of questions addressing particular aspects of the requirements. This list is then supplied to the customer and a Requirements Review meeting is scheduled and held with customer and user, e.g., clerks, agents. During the meeting the questions are discussed to establish a more specific and unambiguous set of requirements.

VV&T Purpose: To produce a requirements specification providing the foundation from which more formal requirements specification, VV&T planning, and test planning will be accomplished.

VV&T Technique: The review itself is the VV&T technique used in this activity. Some of the questions addressed during the review could be:

- o Shouldn't a claims record contain some kind of indication as to the nature of the claim? For example: if it is due to an accident, who was at fault?
- o How is the "reasonableness" of a claim amount determined?
- o How does one know what claim numbers are valid for which agents?
- o When is the premium rate computed? How is it computed?
- o Shouldn't the acceptance criteria include provisions for testing more than just the functional capabilities?

AIMS Requirements

A system called the Auto Insurance Management System (AIMS) is to be developed which will provide an automated set of automobile insurance support capabilities integrated through the use of a common database. The basic capabilities to be provided include: accounts payable, accounts receivable, management report generation, and database management.

System Information: Information contained in the database includes client records, claims records and the payout account. There is one client record for each policy holder and contains:

- o policy number;
- o name and address of the client;
- o agent number;
- o policy effective date and expiration date;
- o name, birth date, sex, marital status, driving record of each driver;
- o vehicle information - make, model, style, year;
- o insurance coverage - comprehensive, medical, collision and deductable, premium rate classification, balance due, date due, credited amount (e.g., from prepaid premium), number of claims made on this policy and total amount paid out.

There is one claims record for each claim and contains:

- o claim number and date of claim;
- o associated accident report number;
- o driver's name;
- o payee (e.g., repair shop), name and address;
- o agent number and policy number of client making claim.

There is one payout account record in the database and contains:

- o account balance;
- o date and time of the last change to the account balance;
- o minimum allowed balance, date and time of last minimum change;
- o maximum allowed balance, date and time of last maximum change;
- o total year-to-date claims and premium total;

Accounts Payable: The accounts payable function processes claims transactions and issues payment checks to the payee which will either be the repair shop or the client. The transactions are input to the AIMS system from a file containing the day's claims. Having been input, a claims transaction is validated by checking the consistency of the client information contained in the transaction with that contained in the client record, by verifying that the claim number is valid for this agent, and by checking the "reasonableness" of the amount. Once validated, the new claims record is entered into the database, the amount of the claim is withdrawn from the payout account and the check is issued (i.e., printed). When the claims withdrawal is made from the payout account the account balance is updated and then compared to the minimum allowable balance. If the new balance is less, then a notice is issued (for management) indicating the situation. The date and time of last change to the account balance is also updated as well as the year-to-date claims total.

Accounts Receivable: The accounts receivable function issues policy notices and processes premium payments. Policy notices are issued by a batch program which runs once per day. The program reads all client records and checks each record to determine if a premium due notice or a cancellation (i.e., past due) notice should be issued and if so, prints the appropriate notice. When a premium payment is received a transaction is entered into a file which is processed daily by the AIMS to update client records with the premium payment. The information included in the transaction includes the policy number, client's name, agent number, due date and amount paid. Once received, the transaction is validated by verifying that the input data is consistent with that in the client record and that the amount paid is sufficient. The client record is then credited with the payment and the amount is deposited in the payout account. The payout account balance, year-to-date premium total, and date and time of last change to the account balance are updated when the deposit is made. Note that the maximum allowable balance is not automatically checked. This is a manual function performed by management.

Management Reports: Four management reports are produced: claims report, new clients report, client profile report and client cancellation report. The claims report is produced on a monthly basis and gives the total number of claims and the total amount paid in claims for a given month. It also performs a trend analysis based on totals from prior months. The new clients report is produced on a monthly basis and lists new clients and their coverage for a given month. The clients are grouped by agent which allows management to view the sales progress of each agent. A company-wide sales trend analysis is also produced. The client profile report provides accident statistics based on driver's age, sex, marital status, etc. This report is produced semi-annually. The client cancellation report lists the clients which were cancelled during a given month and the reason for the cancellation.

Database Management: Database management activities provide for client and payout account management capabilities. Client records are entered, queried, modified and cancelled (i.e., made inactive but not deleted). A log of all client transactions is also stored in the database (or possibly on tape). Payout account functions include query, modification of the balance limits, external deposits and external withdrawals. External deposts and withdrawals are made from/to other company financial resources whenever the account balance exceeds the allowable limits. A log of all deposits and withdrawals (including premiums and claims) is kept on the database.

Acceptance Criteria: The acceptance criteria for the AIMS is the successful execution of a set of acceptance tests. An execution is successful if it correctly performs the desired function within the required execution time. The acceptance tests are specified by an independent team and reviewed by management, user and development personnel.

Figure B.2.1 *Informal prose requirements*

B.2.1.2 Requirements Analysis

The requirements analysis involves translation of the informal prose requirements into a formal representation. This results in identifying other aspects of the requirements needing clarification or further definition. For this example, the graphical representation (GR) scheme used is a modification of the Systematic Activity Modeling Method [SAMM].

VV&T Purpose: To identify inadequately specified requirements such as incomplete, ambiguous, or otherwise unclear requirements statements.

VV&T Technique: Formal reviews are used to achieve the above purpose on this project. Problem issues identified during the requirements analysis are documented and distributed to the customer and a second Requirements Review is scheduled. This review again involves dialogue between the customer and the developers; it centers on the formal requirements statement and the identified issues. The result is a revised set of requirements in both formal and informal forms and a graphical representation (GR). Specific activities performed within this review are:

- o Verification that all requirements have been correctly represented using the formal scheme,
- o Identification of the problems encountered during the restatement elaboration of the requirements, and
- o Discussion and resolution of the problems.

Example:

The formal representation for the basic system and the accounts payable function are shown in figure B.2.2. The graphical representation is interpreted as follows:

Master input files are at the top of the diagram.

Master output files are at the right of the diagram.

The upper half of figure B.2.2 is the root which contains five modules, A–E. The data flow within the root and to and from master files are labeled according to their source. If the data are internal to the root, its identifier is preceded by the module letter.

The lower half of figure B.2.2 is an expansion of module A from the root. The lower left corner of each box contains the parent, i.e., A in the root. The lower right corner of each box is the letter designator for each module, i.e., A–E. Data created by the accounts payable activity labeled according to source, e.g., data B.1, a validated claims transaction, is created by module B, validate claims transaction, and used by modules C–E. Data B.2, invalid claims transaction notice, is created by B and put on master file 7, user/client notices.

Some of the problems which could be identified are:

- o What does the system do with an invalid claims transaction? Solution: Output a notice to the user identifying the errors.
- o The involved driver's record in the client's record needs to be updated to reflect a new claim due to an accident. There does not appear to be enough information in the client record for this. Solution: Add the necessary information to the claims transaction.

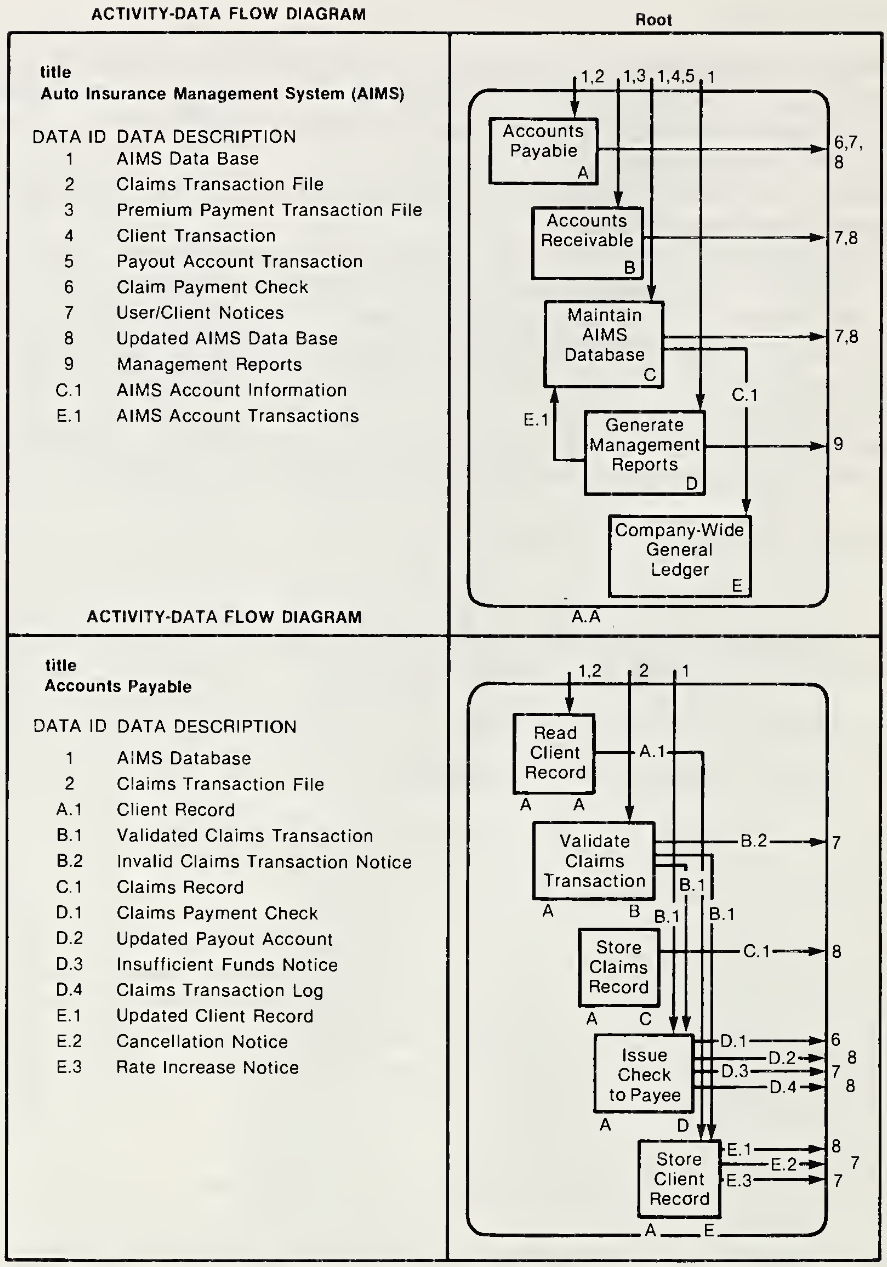

Figure B.2.2 *Requirements graphical representation*

B.2.1.3 VV&T Planning

VV&T planning is one aspect of the overall planning process. It is accomplished in parallel with other planning activities and the requirements identification activity.

VV&T Purpose: To choose VV&T practices which can be implemented to suit the project needs. The objectives are:

- o Identify the goals of the AIMS project's VV&T activities,
- o Select supporting VV&T techniques and tools, and
- o Develop plans for each phase's VV&T activities. (Plans include tasks, e.g., acquiring or developing tools), schedules, responsibilities, and resources.)

B.2.1.4 Initial Test Case Generation

The AIMS requirements will be analyzed and test cases will be designed to test the functional capabilities of the system. These test cases will also form the basic set of acceptance tests.

VV&T Purpose: To design test cases which, when used to test the AIMS software, will maximize the possibility of revealing the presence of errors in the software.

VV&T Technique: Requirements-based functional testing is applied to generate this initial set of test cases.

Example:

In the accounts payable function a claims transaction is validated by checking (among other things) that the claim number is valid for the given agent. Each agent has a specified range in which claim numbers associated with claims issued by that agent must fall. Assuming an agent was assigned claim numbers in the range 801000 to 801999, test cases which are generated to test accounts payable should include claim numbers as follows.

Test data class	*Test claim number*	*Expected output*	*comment*
Non-extremal	801500	None	valid
Non-extremal	801317	None	valid
Extremal	801000	None	upper bound
Extremal	801999	None	lower bound
Extremal	800999	Invalid claim number	lower bound−1
Extremal	802000	Invalid claim number	upper bound+1
Special	80100A	Invalid claim number	
Special	80100Z	Invalid claim number	
Special	80150	Invalid claim number	
Special	-01500	Invalid claim number	
Special	80L500	Invalid claim number	

B.2.2 Preliminary Design Subphase Activity Descriptions

B.2.2.1 Refinement of Graphical Representation

The GR diagrams developed during requirements analysis will be decomposed to reflect the requirements for the system in more detail.

VV&T Purpose: The completeness and consistency of the GR description of the requirements and preliminary design should be ensured.

VV&T Technique: A review of the resulting diagrams will be performed to verify:

- o identification of all basic activities necessary to perform a particular function,
- o identification of all inputs and outputs required by each activity, and
- o consistency and completeness of the data flows.

Example:

Within the accounts payable function, there is no indication as to the action to be taken when a claim transaction is processed for a claim which has been previously entered. This error would be discovered during the review of the GR activity for the accounts payable function.

B.2.2.2 Specify Information Design

The preliminary design of the information consists of a detailed user input/output specification and a description of the basic content and structure of the data used by the system. The detailed user input/output specification essentially amounts to preparing a user's manual for the system. The formats used to input claims and premium payment transactions are defined as well as the output responses. The printed report formats for the management reports are also defined. Specification of the basic data structures and content will consist of identification of variables and records needed by the system, and the relationships among them.

VV&T Purpose: The VV&T purpose in this activity is twofold. First, the detailed user specifications need to be shown to be usable and that they satisfy the needs of the user. Second, the system data structures and content need to be verified and shown to be complete (i.e., that which is required to perform all system functions) and correct (i.e., the data types and relationships are consistent with the functions which need to be performed).

VV&T Technique:

- o A formal session will be held with the customer to review the detailed user input/output specifications. This session will be preceded by informal dialogue between the user community and the developers to assist in the development of the specifications. Once satisfied, the customer will formally sign off on the specification.
- o Formal inspections of the system data structures and content will be performed.

Example:

Discovered by the customer participating in the formal review of the detailed input/output spec was that a client is not always the owner of the car, so that lien-holder information needs to be included in the client record.

B.2.2.3 Design Program Architecture & Allocate Requirements

The program architecture design gives a complete high-level description of the software. It refines and groups functions defining software components and interfaces.

VV&T Purpose: Requirements are cross-referenced by the design to ensure that all the requirements have been addressed.

VV&T Technique: Requirement trace-back.

Example:

A complete set of cross-references is defined and maintained. These show the evolution from the prose requirements to the requirements represented by the GR and finally to the components identified in the design.

B.2.2.4 Design Basic Control Flow

The GR represents the data flow within a system but only shows control flow in an implicit way. The system's control flow, therefore, needs to be explicitly designed. The activities identified in the GR need to be mapped into modules. The control flow between modules must also be described using an informal design language. This defines the program architecture. The hierarchical structure of the modules comprising the system are developed.

VV&T Purpose: To produce a correct and understandable description of the basic control flow of the system.

VV&T Technique: An inspection of the control flow design will be performed to verify:

- o consistency with the GR representation,
- o correctness of the high-level logic, and
- o quality of the modularization, i.e., are the functional boundaries natural?

B.2.2.5 Test Case Generation

VV&T Purpose: To generate test data that will exercise and test each function, and also to demonstrate that the code is consistent with the design.

VV&T Technique: Design-based functional testing.

Example:

> Test cases for a function adding the amount of the premium payment to the payout account would include: a negative (or zero) amount, an amount which is greater than zero but less than that which would leave the balance larger than the maximum allowed, and one which would leave the balance greater than the maximum allowed.

B.2.2.6 Preliminary Design Review

At the completion of the preliminary design activity, a formal review is held. This review involves management and technical staff representing the developer and the customer/user and covers all aspects of the design and results of VV&T activities. Management of customer/user and developer sign off of acceptance is required.

B.2.3 Detailed Design Subphase Activity Descriptions

B.2.3.1 Detailed Database Design

The format and structure of the data to be stored in the system database is designed. This includes describing data which are logically related in the form of records, as well as the relationships existing between records. The logical structure of the database will be described using a graphical database design representation. Record descriptions will be specified in a data definition language. Examples are shown in figures B.2.3 and B.2.4.

In figure B.2.3, ovals represent record access (key) fields, boxes represent records, "1:M" means that for each client record there are potentially many (1 or more) claims records.

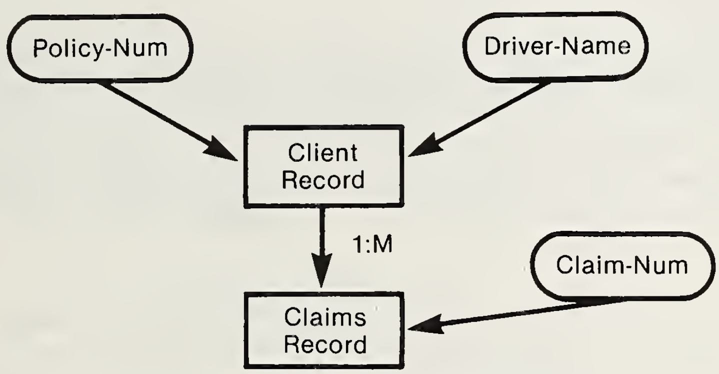

Figure B.2.3 *Sample database schema showing client-claims relation*

VV&T Purpose: The database design must be verified for consistency with the preliminary design. In addition, the database structure will be verified to ensure that it is correct and is reasonable with respect to potential storage consumption and access time.

VV&T Technique: An inspection of the database design is performed to ensure that the above VV&T purpose is met.

Example:

> During the inspection of the database design an error is found in the claims record (fig. B.2.4) where POLICY-NUM is identified as the key field whereas the schema diagram (fig. B.2.3) indicates CLAIM-NUM. The solution is to change the key field in the claims record description to CLAIM-NUM.

```
record name is CLAIMS
 location mode is calc in CALC-KEY using POLICY-NUM
  01    CLAIM-NUM                                    PIC 9 (6)
  01    DATE-OF-CLAIM                                PIC 9 (6)
  01    ACC-REP-NUM                                  PIC 9 (9)
  01    DRIVER
        02 LAST                                      PIC X (15)
        02 FIRST                                     PIC X (15)
        02 MIDDLE-INTL                               PIC X
  01    PAYEE
        02 NAME                                      PIC X (31)
        02 ADDRESS
           03 STREET                                 PIC X (24)
           03 CITY                                   PIC X (15)
           03 STATE                                  PIC X (2)
           03 ZIP                                    PIC X (5)
  01    POLICY-NUM                                   PIC 9 (8)
  01    AGENT                                        PIC 9 (5)
```

Figure B.2.4 *Sample CLAIMS record description*

B.2.3.2 Detailed Module Design

Detailed module design includes, for each module, a description of the function performed and descriptions of input and output data, as well as a high-level description of how the function is to be done (i.e., the algorithm used).

VV&T Purpose: To show that (1) all of the system's functional capabilities are addressed by one or more modules, and (2) each module addresses one or more system functions. Moreover, relationships among and interfaces between all modules are identified and verified.

VV&T technique:

- o Inspections of the system modules include (1) manual checking of the module interfaces to ensure that all modules are used and that their inputs and outputs are consistent, and (2) informal verification of the correctness of the algorithms used.
- o Requirements tracing is accomplished by identifying each module with the lowest level GR activity (from the preliminary design) in which the module is contained.

Example:

> A module which updates the date and time of the last access to the payout account record has the premium payment transaction as one of its inputs. However, manual interface checking detects an inconsistency whereby the premium payment transaction is not supplied. As it turns out, the transaction is not used within the module and is deleted as an input.

B.2.3.3 Test Case Generation

This involves refining and adding to test data previously developed.

VV&T Purpose: Test cases are developed to exercise and test the internal structures and functions of modules.

VV&T Technique:

- o Branch testing
- o Path testing

Example:

> The module which validates a claim number checks for six error conditions. Associated with these conditions are three actions. Test data are developed to exercise all combinations of error conditions and resulting actions, i.e., all branches and all paths through the modules.

B.2.3.4. Design Review (DR)

After the detailed design is completed, a formal review is held. Primarily involving project management and technical personnel, this review covers all aspects of the design (including the test cases). Sign off by management indicating their acceptance of the design is required.

B.2.4 Programming Subphase Activity Descriptions

B.2.4.1 Code Development

The detailed design of a given component provides the information needed to write the code for that component in the host programming language, e.g., COBOL. Once written, the code is entered into the computer and all compilation errors are removed.

VV&T Purpose: VV&T of the compiled code is performed to:

- o Verify the consistency of the code with the detailed design.
- o Identify errors, and
- o Ensure adherence to programming standards.

VV&T Technique: Inspection of each system module.

Example:

> During an inspection of "issue policy notices" module the section of code responsible for issuing a premium due notice is found to be in error. The error is that the premium due notice is printed without having the appropriate data moved into the printer buffer. A sample portion of the inspection checklist used is shown below in figure B.2.5. This particular error is discovered using question two under "data reference."

DATA DECLARATION

1. Are all variables declared?
2. Are the correct attributes assigned?
3. Are variables properly initialized?
4. Are variable naming conventions followed?
5. Is the proper explanatory comment included for each variable?

DATA REFERENCE

1. Are there any unreferenced variables?
2. Are there any references to unassigned variables?
3. Are subscripts within range?
4. Are there off-by-one errors in subscript computations?

•
•

Figure B.2.5 *Sample portion of code inspection checklist*

o A cross-referencer is used to produce cross-reference lists of all identifiers used by a program. This list is included with the source code listings for module inspections.

Example:

> A careful examination of the cross-reference listing of module ISSUE-CHECK in ACCOUNTS-PAYABLE during the code inspection indicated that two variables, PAYOUT-ACCOUNT-BAL and PAYOUT-ACCT-BAL, were referenced. The error was that PAYOUT-ACCOUNT-BAL, should have been coded "PAYOUT-ACCT-BAL."

B.2.4.2 Module Testing

An incremental, bottom-up testing strategy is used to test the AIMS modules. This involves individually testing the lowest level modules; then combining and testing those modules with the higher level modules which call them. The process continues until all modules are combined into the complete system. Test drivers are written to control the testing of the individual modules. The test data used is that created by design-based functional testing which were generated from analyses of the functional, structural and interface specifications of the individual modules during detailed design.

VV&T Purpose: To reveal errors present in the individual modules.

VV&T Technique: A test coverage analyzer is used to supplement module testing. Each module to be tested is instrumented to collect execution frequency counts and then executed. The execution counts for each statement are then listed with the corresponding statement by a post-execution routine. Untested or poorly tested portions of the module can be identifed and additional test cases can be generated to test those specific segments.

Example:

> ACCOUNTS-PAYABLE processes claims transactions read from a file which contains a given day's claims. The module contains a check to verify that each record is indeed a claims transaction and, if not, invokes an error handling routine which logs the error. Use of a test coverage analyzer showed that this particular situation did not arise during testing of the module using the tests created during detailed design. As a result, those tests are supplemented with invalid claims transactions and the module retested. This, in turn, results in an error being revealed whereby the error handler responds with an incorrect output response.

B.2.4.3 Function Testing

Function testing of AIMS uses the test cases developed from requirements-based functional testing during preliminary design to test the functional capabilities of the AIMS software.

VV&T Purpose: To reveal errors where the software fails to perform a function as specified in the requirements.

Function testing is supplemented with the use of a file comparator. Associated with each of the requirements-based functional test cases is the expected output. This is stored on a file in the exact format expected to be produced. When the AIMS software is tested, the resulting output is stored on a separate file. A file comparator is used to detect automatically any discrepancies which may have occurred.

Example:

> In preparing the test cases for the New Clients report, a form is used which formats the expected output data in accordance with the specification. Each report corresponding to a given test case is then stored on a file in the order in which the tests are to be executed. Testing is then performed and the actual output is compared to the expected output using a file comparator. The results show the presence of two errors, a format error and a data output error. The format error is a misalignment caused by incorrect spacing between output fields. The data output error is a missing agent name which is to be printed with the agent number.

B.3 EXAMPLE 2: Software Development Using a Comprehensive VV&T Approach

The comprehensive VV&T includes those techniques contained in the basic approach described earlier as well as those described in this section. The additional tools and the applicable lifecycle phase are shown below.

- o Preliminary Design
  - Assertion generation
- o Detailed Design
  - Assertion generation

- o Code
  - Interface checker
  - Data flow analyzer
  - Assertion processor
  - Standards analyzer
  - Requirements trace-back

Table B.2 *Example 2. Summary software development using a comprehensive VV&T approach*

Subphases	Requirements	Preliminary design	Detailed design	Programming
INPUT	•(No additions to basic approach)	•(No additional inputs)	•Preliminary design document including assertions	•Detailed design document including assertions
OUTPUT	•(No additions to basic approach)	•Preliminary design document including assertions about the design	•Detailed design document including additional assertions	•(No additional outputs)
SUPPORTING TECHNOLOGY	•(No additions to basic approach)	•Assertion generation	•Assertion generation	•Interface checker •Data flow analyzer •Assertion processor •Standards analyzer •Requirements traceback
ACTIVITIES	•(No additions to basic approach)	•Design basic control flow	•Detailed module design	•Code development •Module testing

B.3.1 Requirements Subphase Activity Description

(No additions to basic approach.)

B.3.2 Preliminary Design Subphase Activity Description

VV&T Technique: Assertion generation is used to specify the desired functional properties of the individual modules. This is done by including in the module specifications input and, to the extent possible, output assertions.

Example:

Policy numbers are stored in the database in blocks of arrays where each block contains a fixed number (n) of policy numbers (policy-num) and the address (policy-addr) of their associated client records. Policy numbers are stored in the policy-num array in ascending order. A procedure, find-policy, is called to search the policy-num array for a supplied policy number and return the address of its client record. If the supplied policy number is not found an address of zero is returned. The input and output assertions which capture the functional properties of find-policy are given below.

1) /**assert input* policy-num (1)< = num< = policy-num (m) */ *and*
2) /**assert input forall* i *in* 1...n–1:policy-num (i)< = policy-num (i+1) */
3) /**assert output* (*exists in* i *in* 1...n : num = policy-num (i)) */ *or*
4) /**assert output* (add=0 *and forall* i *in* 1...n:num = policy-num (i)) */

B.3.3 Detailed Design Subphase Activity Description

VV&T Technique: Assertions are generated to include algorithmic detail in addition to input and output specifications of the functional properties of the individual modules.

Example:

The example in the previous section describes the find-policy procedure and specifies the input and output assertions associated with it. Shown in figure B.3.1 is the PDL for find-policy which is implemented using a binary search algorithm.

The input and output assertions capture the functional properties of the procedure independent of the algorithm used to implement the search. Assertions 1, 2 and 3, however, capture conditions which are very dependent upon the algorithm. Assertion 1 is always correct whenever num is in the policy-num array. If num is not in the array, assertion 1 is violated the last time through the loop (when high = low). This is an acceptable result, however, in that num should be a valid policy number.

Find-policy:

```
  /* searches sorted global array policy-num for
num (input argument) and, if
   found, returns the associated policy-addr in
addr (output argument). If
   not found a zero is returned in addr */

   /* assert input policy-num (1)< = num< =
policy-num (n) */
   /* assert input forall i in 1...n-1: policy-num
(i)< = policy-num (i+1) */

   set addr to 0
   set low to 1
   set high to n

   do until high< low or num = policy-num (i)
(1) /* assert 1< = low< = high< = n and policy-num
(low)< = num<=
       policy-num (high) */
   set m to (low + high) /2
   if num< policy-num (i)
   set high to m-i
   else if num> policy-num (i)
   set low to m+i
   else goto successful
  enddo

   /* unsuccessful */

(2) /* assert high = low-1 and policy-num (high)<
num< policy-num (low) */
   /* assert output addr = 0 and forall i in
1...n: num ≠ policy-num (i) */
   return

   /*successful*/

   set addr to policy-num (i)
(3) /* assert 1< = low< = m< = high< = n and num =
policy-num (m) */
    /* assert output exists i in 1...n: num =
policy-num (i) */
   return

  end find-policy;
```

Figure B.3.1 *Detailed PDL with ASSERTIONS*

B.3.4 Programming Subphase Activity Descriptions

B.3.4.1 Code Development

The code development activities described in earlier sections are supplemented in a full tool set environment with an interface checker, data flow analyzer, and standards analyzer. These tools can be separate but are often included as capabilities provided by a single tool. They are all static analysis techniques and are therefore applied prior to software testing. The output resulting from each of the capabilities is included with the material for the formal code inspections.

VV&T Techniques:

o Interface checking is used to check the consistency of the interfaces between modules.

Example:

An error is detected between the module which reads client records for premium payment processing and the "find-policy" module. It is an inconsistency in the type of the arguments for the policy numbers. "Find-policy" is being called with a policy number of type character where it should be type integer.

o Data flow analysis is used to identify variable reference/definition anomalies.

Example:

When data flow analysis is performed on the module which updates the payout account with a premium payment, a reference to an uninitialized variable is noted. The variable should contain the current date and time and is used to update the date and time of the last change to the payout account. A call to the routine which updates the time and date should be made prior to the reference.

o Standards' analyzers are used to ensure adherence to program coding and documentation standards. One of the primary capabilities provided by most commonly available standards' analyzers is the notification of the use of nonstandard language features.

Example:

One of the requirements for the AIMS software is that it be portable. To assist in the development of portable code, a COBOL standards' analyzer is used. All places where a standards' violation occurs is either changed or justified. Even trivial nonstandard features such as the use of the abbreviation "DISP" for "DISPLAY" are detected. In addition, a variety of undesirable standard language constructs such as the "ALTER" statement and "NEXT SENTENCE" clause are detected with the tool.

o Requirements trace-back, via code to design and design to code is used to verify that the code adheres to, and satisfies, the requirements as specified by the design. Both missing code and extraneous code may be discovered.

Example:

One of the requirements for the client record for each policy holder is to contain the number of claims made on this policy. During a trace of the design to the code, it is found that no code exists to keep track of the number of claims. However, code is discovered that keeps track of the number of changes to the coverage.

B.3.4.2 Module Testing

The module testing activities described in earlier sections are supplemented with a dynamic assertions processor. This processor is generally included as part of a broader dynamic analysis tool including, for example, statement execution counts.

VV&T Technique: Assertions processor: A dynamic assertions processor translates assertions, usually specified as part of the source program, into source language statements which check the validity of the assertion during program execution. Generally, when an assertion is violated, an informative message is output.

Example:

Figure B.3.2 shows a portion of a FORTRAN implementation of the find-policy routine from figure B.3.1. Also shown is an example of an assertion violation message which was printed when the assertion in line 14 of the program was violated (i.e., false) during program execution. Subsequent analysis of the problem indicated that the error was an incorrect coding of line 18 from the PDL where HIGH should have been set to M–1, not M+1.

```
                 .
                 .
                 .
13     100  CONTINUE
14 C*           ASSERT(1.LE.LOW.AND.LOW.LE.HIGH.AND.HIGH.LE.N
15 C*           .AND.POLNUM(LOW).LE.NUM.AND.NUM.LE.POLNUM(HIGH))
16              M = (LOW + HIGH)/2
17              IF ( NUM .LT. POLNUM(M) ) THEN
18                  HIGH = M + 1
19              ELSE IF ( NUM .GT. POLNUM(M) ) THEN
20                  LOW = M + 1
21              ELSE
22                  GO TO 200
23              ENDIF
24              IF(HIGH.LE.LOW.AND.NUM.NE.POLNUM(M)) GO TO 100
                 .
                 .
                 .

*** ASSERTION VIOLATION AT LINE 14 OF SUBROUTINE FNDPOL:
    CURRENT EXECUTION COUNT = 2
    LOW = 1, HIGH = 65, N = 64, NUM = 22707,
    POLNUM(LOW) = 16747, POLNUM(HIGH) = 36757
```

Figure B.3.2 *Find-policy subroutine and corresponding assertion violation message*

*U.S. GOVERNMENT PRINTING OFFICE : 1984 0-432-178/521

NBS TECHNICAL PUBLICATIONS

PERIODICALS

JOURNAL OF RESEARCH—The Journal of Research of the National Bureau of Standards reports NBS research and development in those disciplines of the physical and engineering sciences in which the Bureau is active. These include physics, chemistry, engineering, mathematics, and computer sciences. Papers cover a broad range of subjects, with major emphasis on measurement methodology and the basic technology underlying standardization. Also included from time to time are survey articles on topics closely related to the Bureau's technical and scientific programs. As a special service to subscribers each issue contains complete citations to all recent Bureau publications in both NBS and non-NBS media. Issued six times a year. Annual subscription: domestic $18; foreign $22.50. Single copy, $5.50 domestic; $6.90 foreign.

NONPERIODICALS

Monographs—Major contributions to the technical literature on various subjects related to the Bureau's scientific and technical activities.

Handbooks—Recommended codes of engineering and industrial practice (including safety codes) developed in cooperation with interested industries, professional organizations, and regulatory bodies.

Special Publications—Include proceedings of conferences sponsored by NBS, NBS annual reports, and other special publications appropriate to this grouping such as wall charts, pocket cards, and bibliographies.

Applied Mathematics Series—Mathematical tables, manuals, and studies of special interest to physicists, engineers, chemists, biologists, mathematicians, computer programmers, and others engaged in scientific and technical work.

National Standard Reference Data Series—Provides quantitative data on the physical and chemical properties of materials, compiled from the world's literature and critically evaluated. Developed under a worldwide program coordinated by NBS under the authority of the National Standard Data Act (Public Law 90-396).

NOTE: The principal publication outlet for the foregoing data is the Journal of Physical and Chemical Reference Data (JPCRD) published quarterly for NBS by the American Chemical Society (ACS) and the American Institute of Physics (AIP). Subscriptions, reprints, and supplements available from ACS, 1155 Sixteenth St., NW, Washington, DC 20056.

Building Science Series—Disseminates technical information developed at the Bureau on building materials, components, systems, and whole structures. The series presents research results, test methods, and performance criteria related to the structural and environmental functions and the durability and safety characteristics of building elements and systems.

Technical Notes—Studies or reports which are complete in themselves but restrictive in their treatment of a subject. Analogous to monographs but not so comprehensive in scope or definitive in treatment of the subject area. Often serve as a vehicle for final reports of work performed at NBS under the sponsorship of other government agencies.

Voluntary Product Standards—Developed under procedures published by the Department of Commerce in Part 10, Title 15, of the Code of Federal Regulations. The standards establish nationally recognized requirements for products, and provide all concerned interests with a basis for common understanding of the characteristics of the products. NBS administers this program as a supplement to the activities of the private sector standardizing organizations.

Consumer Information Series—Practical information, based on NBS research and experience, covering areas of interest to the consumer. Easily understandable language and illustrations provide useful background knowledge for shopping in today's technological marketplace.

Order the **above** *NBS publications from: Superintendent of Documents, Government Printing Office, Washington, DC 20402.*

Order the **following** *NBS publications—FIPS and NBSIR's—from the National Technical Information Service, Springfield, VA 22161.*

Federal Information Processing Standards Publications (FIPS PUB)—Publications in this series collectively constitute the Federal Information Processing Standards Register. The Register serves as the official source of information in the Federal Government regarding standards issued by NBS pursuant to the Federal Property and Administrative Services Act of 1949 as amended, Public Law 89-306 (79 Stat. 1127), and as implemented by Executive Order 11717 (38 FR 12315, dated May 11, 1973) and Part 6 of Title 15 CFR (Code of Federal Regulations).

NBS Interagency Reports (NBSIR)—A special series of interim or final reports on work performed by NBS for outside sponsors (both government and non-government). In general, initial distribution is handled by the sponsor; public distribution is by the National Technical Information Service, Springfield, VA 22161, in paper copy or microfiche form.

U.S. DEPARTMENT OF COMMERCE
National Technical Information Service
5285 Port Royal Road
Springfield, Virginia 22161

OFFICIAL BUSINESS

POSTAGE AND FEES PAID
U.S. DEPARTMENT OF COMMERCE
COM-211

3rd Class Bulk Rate

U.S.MAIL