Eldon Y. Li is an associate professor and the coordinator of the management information systems program at the School of Business, California Polytechnic State University, in San Luis Obispo, CA.

Journal of Systems Management, 41 (8), August 1990, 23-31

Software Testing In A System Development Process: A Life Cycle Perspective

BY ELDON Y. LI, Ph.D.

Computer software is a major component of an information system (IS) whose reliability is critical to the performance of an organization. The reliability of an IS has four facets people, hardware, software and data. Among these four facets, the reliability of software is a joint responsibility of computer science and information system professionals; the former handles technical software and the latter application software. Regardless of the types of software involved, the reliability of software must be achieved through a continuous effort of planning, analysis, design, programming, testing, installation and maintenance. To quote an old sage, "Quality is built in, not added on." Studies have shown that most of the system errors detected during the testing phase originate in the early analysis phase.[2] Therefore, software testing should start early in the system development process.

A system development process today may come in many different forms. Depending on the structure of the intended system, a system development life cycle (SDLC) process[6] or a prototyping process [2,7,13] or a mixture of both[4] can be followed. An SDLC process typically is applied to a system with clear requirements definitions, well structured processing and reporting, and a long and stable life expectancy. On the other hand, a prototyping process is suitable for a system of ambiguous or incomplete requirements with ad hoc reporting capability, or of a transient life span with ever-changing requirements. It may be used along with an SDLC process to shorten the development time required by a project adopting the SDLC process. The scope of this article is to discuss the software testing process and test activities in a large system project following the SDLC approach. The test process and activities discussed herein is complete and rigorous. Its principles can be applied to a project of any size and using any system development approach,. be it an SDLC a prototyping, ware acquisition or a spiral development and enhancement[4] approach.

The System Development Life Cycle

The SDLC typically consists of three major stages: definition, development and installation, and operation.[6] According to the SDM/70 methodology (a proprietary system development methodology developed by Atlantic Software, Inc. in conjunction with Katch and Associates), these three major stages can be divided into 10 phases. The definition stage contains three phases: service request/project viability analysis (*SR/PVA*), system requirements definition (*SRD*) and system design alternatives (*SDA*). The development stage contains four phases: system external specifications (*SES*), system internal specifications (*SIS*), program development (*PD*) and testing (*TST*). The TST phase is actually interspersed throughout the entire SDLC. Finally, the installation and operation stage begins with a conversion (*CNV*) phase, then implementation (*IMPL*) and ends with a post-implementation review/maintenance (*PIR/MN*) phase.

Figure 1 exhibits the 10 SDLC phases in the SDM/70 and the objective of each phase. Notice that there could be some iterations within SDLC phase or between any two adjacent phases during the definition and development stages. But no feedback is allowed across two or more | bugging is to identify the type and location of phases. Only when an SDLC phase is completed and signed off can the next phase proceed. For example, within the SRD phase in Figure 1, users and systems analysts may go through several changes before the final SRD document is completed. Once the SRD document is signed off, the phase is said to be completed and thus the SDA phase is begun. When the SDA phase is started, it is assumed that the SRD document contains no "major" errors. Although touch-up changes in the SRD document during the subsequent phases are expected, any major changes

in the SRD are strictly controlled and discouraged. The word "major" implies that the change can cause a substantial overhaul of all its subsequent phase documents--a phenomenon commonly called "the ripple effect." This practice is applied to all SDLC phases, making the SDLC model also known as the "linear" or "waterfall" model. Because the model requires fully elaborate documentation as completion criteria for each phase, it is also called a "document-driven" model.[4]

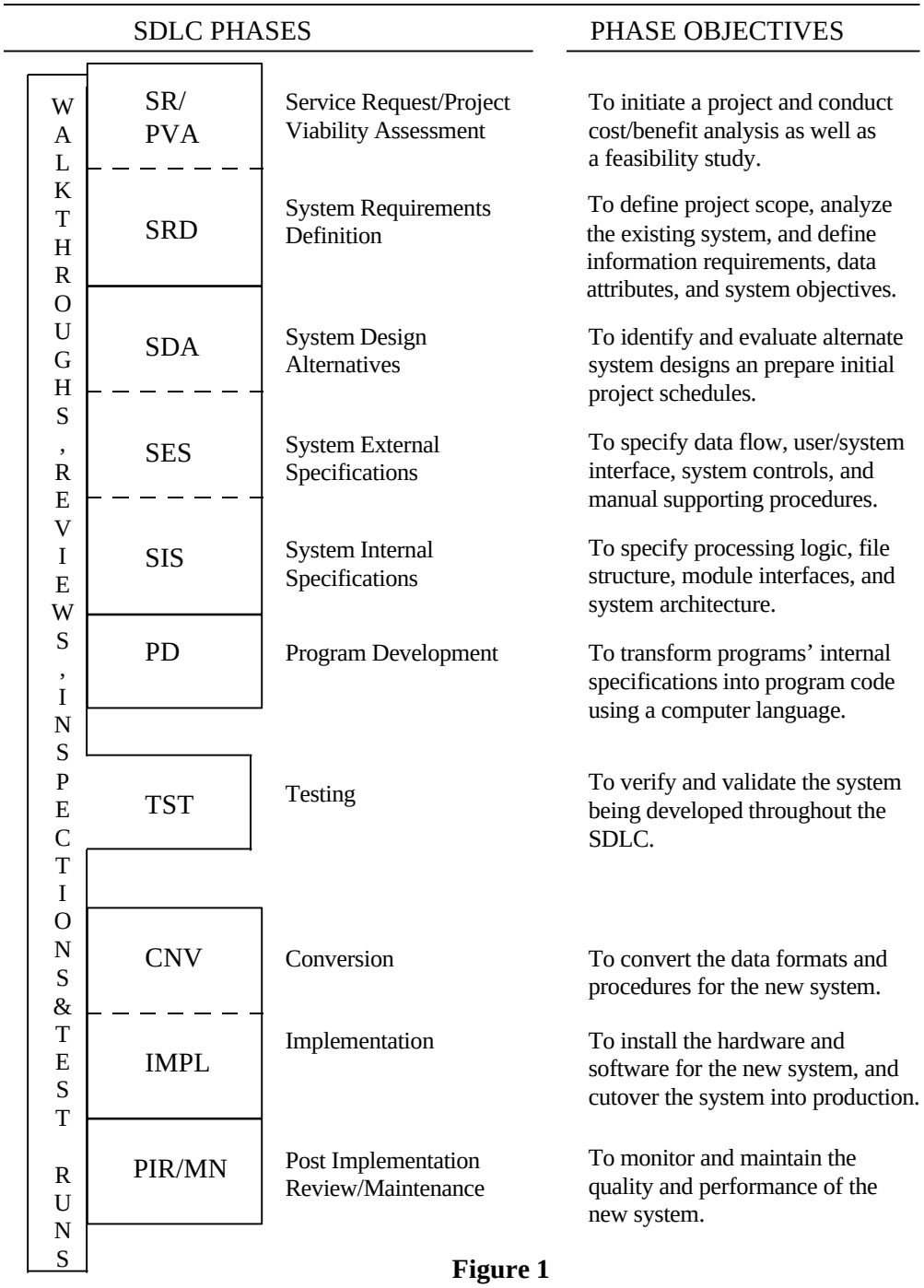

Figure 1
The Ten Phases of the System Development Life Cycle

The rationale of the model in Figure 1 was based on the dreadful experience with the tremendous time and cost incurred by a "major" change in the document of an earlier SDLC phase. In retrospect, such stringent change control was very popular before 1980 because there was no easy-to-use tool commercially available then that allowed easy modification of system contents (such as designs, code and documentation). Today, however, the IS market is rich with computer-aided software engineering (CASE) tools, automated code generators, automated code and document management systems, and other automated tools. Such tools have made last-minute changes possible and easier. They make the spiral development process described by Boehm[4] become a reality.

Software Testing In The SDLC Process

Software testing should not be confused with software debugging. The former is a process of finding "unknown" errors whereas the latter a process of removing "known" errors. In general, the objective of testing is to find errors in a software and to see not only if this software does not do what it is supposed to do (i.e. a deficiency), but if it does what it is not supposed to do (i.e. a malfunction). In contrast, the objective of debugging is to identify the type and location of the "found" error and subsequently remove it by a re-definition, re-design or re-code, depending on the level of testing through which the error was found. Thus, testing can be viewed as a destructive process and debugging as a constructive one. However, testing and debugging should go hand-in-hand and be performed one right after another.

Regardless of the type of development process one may adopt, testing should start as soon as a project is launched. Figure 2 shows the software testing activities throughout the SDLC. At the first five phases of the SDLC (before any module coding is completed), manual testing techniques such as structured walkthroughs,[l5] desk checking, reviews5 and inspections1 5 are used to verify that the end products of these SDLC phases are correct based upon the output of their prior phases. For example at the end of the SRD phase, the SRD document is verified by comparing it to the service request and user's current opinions. If any mistakes or necessary changes are discovered, they should be fed back to the SRD process and the SRD process should be reiterated. If no errors or changes are found, the SRD documents are then approved or signed off by the user and the next phase (the SDA phase) then proceeds.

Once the program development (PD) phase begins its course, manual testing should be performed before and after a program is subjected to a computer-based test. The purpose of such manual testing is to ensure that the programs being developed in this phase confirm to the System Internal Specifications completed in the last phase. After the program development (PD) phase, manual testing is performed to ensure that the current phase documents were all consistent with the end products of the prior phases along with the feedback from the end user. For example, in the implementation (IMPL) phase, implementation plan, user's guide and operations guide should be walked through, reviewed or inspected to see if they conform to the system requirements definition and the system external specifications as well as the feedback from the end user. Note that the involvement of the end users at the early phases such as SRD, SDA and SES is the key factor leading to the successful certification of system completion.

Levels Of Computer-Based Testing

As alluded to earlier, computer-based testing can only be performed when the coding of a program module is completed during the program development (PD) phase. A program should be desk-checked and free of compilation error before its code is manually reviewed or inspected. It is after such manual testing that a computer-based test should proceed.

Computer-based testing can be classified into seven levels according to its testing objectives, each of which focuses on a particular class of errors. The following is a discussion of these levels of testing and their corresponding test objectives. Note that the first two levels of computer-based testing (i.e., module testing and integration testing) are also known as "development testing."[16]

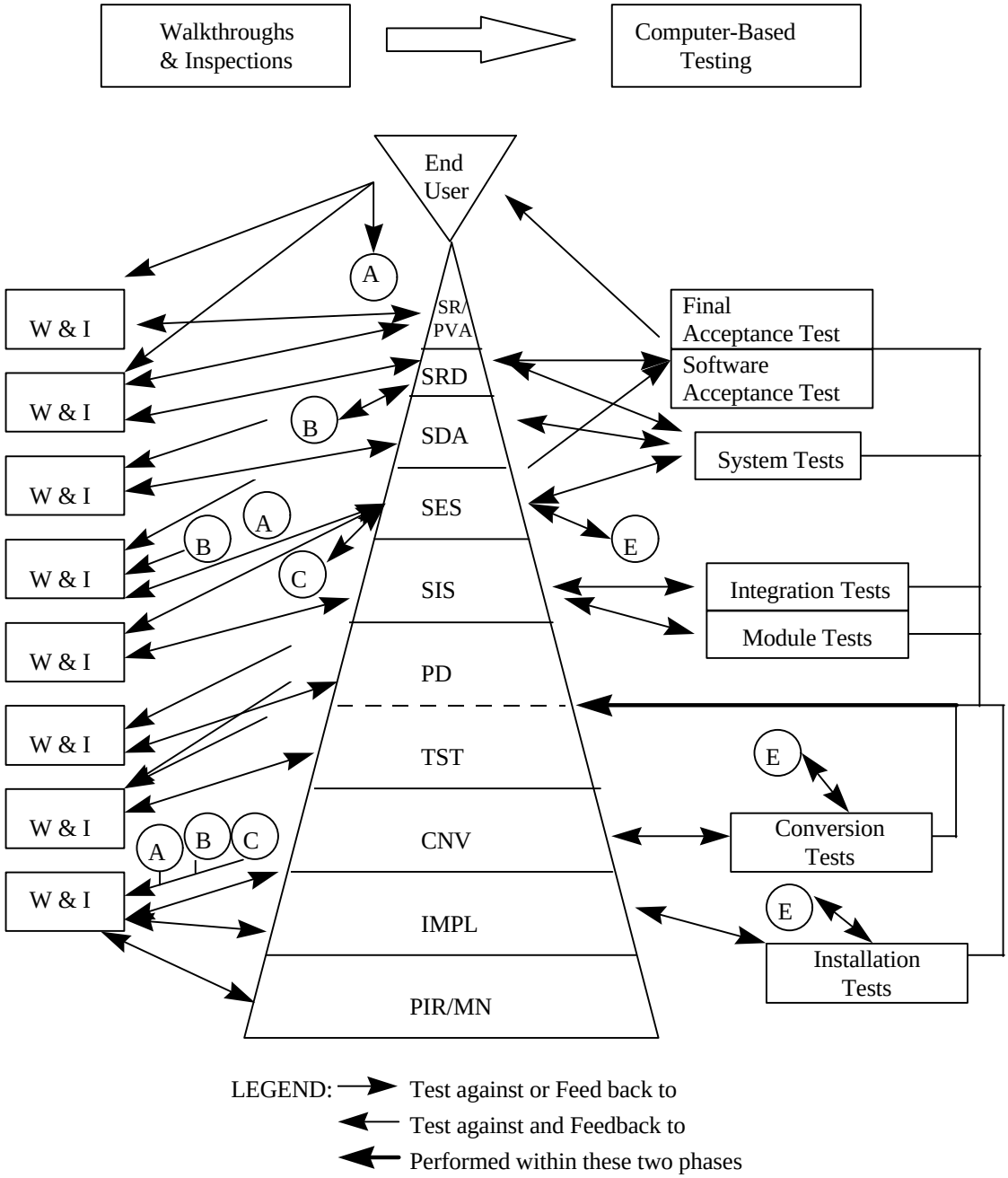

Figure 2.
Software Testing in the
SDLC of a Large System Project

Module (Unit) Testing: A *module* is any collection of executable program statements that can be called from any other module in the program and has the potential of being independently compiled.[11] A module, which can be a closed subroutine, a procedure or a subprogram, is the primary unit of a program's structure. Module testing (or unit testing) is, therefore, a process of testing the individual subprograms, subroutines or procedures in a program. Its purpose is to check if the module contradicts the system internal specifications.l2 Module testing is

performed during the program development phase. Different modules can be tested in isolation simultaneously. Testing a module is much less complex than testing a program/system as a whole. Module testing makes the debugging tasks much easier because it only involves a single module and, when an error is found, the error should be located within that particular module. The person who is testing a module should be the one who knows the internal details of that module -- namely, the programmer who has coded the module. The test cases and data should be derived from the program source code or the system internal s7pecifications (SIS).

Integration Testing: Also called interface testing, incremental testing or string testing, integration testing is a process of merging and testing program modules to see if they can work correctly as a whole without contradicting the system's internal and external specifications. Such integration can be of three levels: program integration, subsystem integration and system integration. While program integration interconnects modules, subsystem integration interconnects programs, and system integration interconnects subsystems.

During an integration test, driver modules and/or stub modules must be created depending on whether top-down or bottom-up integration strategy is being applied. Driver modules are the upper-level modules that must be coded to transmit test data through the lower-level module under test. Stub modules are the lower-level modules that must be coded to simulate the function called by the upper-level module being tested. While top-down testing merges and tests program modules from the upper to lower level, bottom-up testing merges and tests program modules from the lower to upper level. In top-down testing, regardless of the module level, a real module (as opposed to a sub module) can be integrated into the program only when at least one of the upper-level modules calling it has been tested. In contrast, in the bottom-up testing, a real module (as opposed to a driver module) can be integrated only when all of the lower-level modules it calls have been tested. Usually, the order of test progression can be derived from the structure of the program or system being tested.

Both top-down and bottom-up integration strategies can be applied to each level of integration. Relatively speaking, bottom-up testing performs well in testing small or medium size systems, while a mixture of both top-down and bottom-up testing is recommended for testing large-scale systems. All three levels of integration tests require the tester to know the internal structure as well as the external interfaces of the program, subsystem or system under test. The test cases and data should be derived from the system's internal and external specifications.

System Testing: During the integration testing process, the goal is to explore the incompatibility of interface between the system components. Once the system integration testing is completed, the focus is on verifying that the system as a whole is structurally and functionally sound. A battery of structured and functional tests must be performed to attain the system test objective. These tests and their respective purposes are listed in Table 1. While the structural tests cases and data are designed by analyzing the system objectives (specified in the system requirements definition) and then formulated by analyzing the user documentation (whose contents are based on the system design alternatives and the system external specifications), the functional test cases and data are derived from the system requirements definition and the system external specifications.

System testing requires the tester to know the functions and the inter-components interface of the system under test; it does not require the tester to know the internal structure of that system. An excellent example of system testing has been given by McCabe and Schulmeyer. 'ø Clearly, designing test cases and data to test a large system in its entirety is an uneasy task. We highly recommend the tester to use parallel processing, if possible, because through parallel processing the tester can use real-life data as the test data and the test results of the new system can be easily verified by the actual results of the old system.

Software Acceptance Testing: As soon as system testing is completed, software acceptance testing is performed. The newly developed software system is executed in as near the user's operational environment and host computer configuration as possible. Therefore, it is also known as an "operational demonstration."l6 The purpose of this process is to ensure that the software system meets the previously defined system external specifications, acceptance criteria and system requirements definition before it is installed, integrated and checked out in the operational environment.

Software acceptance testing should be performed by the system's end users or a group of test specialists with assistance of the development group. The test cases and data should be derived from the system external specifications and the acceptance criteria (defined in the project test plan during the system external specification phase). Once the software system has been accepted by the user, formal certification of the system should be issued. The system is then ready for file conversion, software installation, system integration and final checkout in the full operational configuration.

Table 1
Structural and Functional System Tests

TYPE OF TEST	TEST PURPOSE
Structural system tests:	
Compliance test	Is the system developed in accordance with standards and procedures?
Configuration test	Does the system work under minimum and maximum configurations?
Documentation test	Is the user documentation accurate?
Maintainability test	Is the system easy to maintain and the internal logic documentation accurate?
Operations test	Can the system be executed in a normal operational status?
Performance test	Does the system achieve desired levels of performance or efficiency under certain workload and configuration conditions?
Portability test	Is the system compatible, installable and maintainable?
Recovery test	Can the system be returned to an operational status after a failure?
Reliability test	Does the system meet the reliability objective?
Security test	Is the system protected in accordance with its level of importance to the organization?
Storage test	Does the system have enough main and secondary storage?
Stress test	Can the system perform with peak load over a short span of time?
Volume test :	Can the system perform with heavy load over a long span of time?
Functional system testing:	
Auditability test	Does the system provide audit trails and meet other auditing standards?
Control test	Do the controls reduce system risk to an acceptable level?
Error-handling test	Can errors be prevented or detected and then corrected?
Inter-systems test	Can data be correctly passed from one system to another?
Manual support test	Does the people-computer interaction work?
Parallel test	Are there unplanned differences between the old and the new systems?
Regression test	Do the unchanged system components still perform correctly with the changed component?
Requirements test	Does the system perform as specified?
Usability test	Are the system and documentation user-friendly?

Source: Adapted from Myers[12] and Perry [14]

Conversion Testing: This is a process of testing the file conversion program to see that (1) its program logic meets the specification, (2) the program contains all necessary control procedures and (3) the conversion process yields the expected results. The testing process can be of two levels. The first level of testing is identical to module testing, since the test cases are derived from analyzing the conversion programs' logic supplemented by their internal specifications. On the other hand, the second-level testing is similar to system testing in that the test cases are derived from analyzing the system external specifications and the conversion plan, and should check that

(1) the system has been properly converted, (2) all required files have been converted and have the correct contents and formats, (3) all reports have the desired contents and formats and (4) all control procedures have been properly defined.

Installation Testing: The objective of the installation testing is not to find software errors but to find installation errors. The test cases should check to ensure that (1) the compatible set of system options has been selected by the user, (2) all parts of the system exist, (3) all programs have been properly interconnected, (4) all files have been created and have the necessary contents and (5) the hardware configuration (including any new installation) is appropriate.l2 Therefore, installation testing may include a hardware acceptance test if new hardware has been installed.

Installation test cases can be designed by analyzing the system external specifications and then formulated by analyzing the installation plan. The test should be performed by the development group that is familiar with the characteristics of not only the software but the hardware being installed.

Final Acceptance Testing: This is the final test process before the system is transferred into production (i.e., the start of the post-implementation review/maintenance phase). It is the process of comparing the system to its initial requirements and the current needs of its end users. The test should be performed by the system's end users or a group of test specialists with assistance from the development group. The test cases can be designed by analyzing the system requirements definition and then formulated by analyzing the acceptance criteria and the system external specifications. Many of the test cases developed for the earlier software acceptance testing may be used again here.

Depending on the complexity of the test, parallel processing may be performed to verify the system against the acceptance criteria. Once the system has been accepted by the user, formal certification of the system should be issued and the system cut-over then started. During the cut-over process, a thorough project review is conducted, programming and test documents are handed over to the maintenance group, and the system is transferred from the development environment to the production environment. After the system is completely cut-over, it is then formally in production and the development project is terminated. From now on, any change or enhancement of the system shall be handled by the maintenance group rather than the development group.

Table 2
Test-Case Design Techniques and Their Base Documents at Each Level of Computer-Based Testing

LEVEL OF TESTING	BASE DOCUMENTS	SUITABLE TEST-CASE DESIGN TECHNIQUES
Module (Unit) Testing	System Internal Specs., program source code	White-box supplemented with black-box techniques.
Integration (Interface) Testing	System Internal Specs., System External Specs	White-box supplemented with black-box techniques.
System Testing:		
Functional tests	System Req. Definition, System External Specs	Black-box techniques.
Structural tests	System Req. Definition, System Design Alterna., System External Specs.	Black-box techniques.
Software Acceptance Testing	System External Specs., acceptance criteria, System Req. Definition.	Black-box techniques.
Conversion Testing		
First level	Conversion programs' logic and internal specs	White-box supplemented with black-box techniques.
Second level	System External Specs., Conversion Plan	Black-box techniques.
Installation Testing	System External Specs., Installation Plan	Black-box techniques
Final Acceptance Testing	System Req. Definition, acceptance criteria, System External Specs.	Black-box techniques.

Test-Case Design Techniques

Designing test cases for each level of computer-based testing is surely not an easy task. The techniques for designing test cases to test a program or a system may be classified into *white-box techniques and black-box techniques* [12]. White-box techniques (also called structured, code-based or logic-driven techniques) require the tester to examine the internal structure of the program and derive the test cases and the data from the program logic described in the system internal specifications (SIS) or the program source code. On the contrary, black-box techniques (also called functional, specification-based, data-driven or input/output-driven techniques) do not require the tester to know the internal structure of the program. The test cases and data are derived solely from the system requirements definition or the system external specifications (SES). Recently, Li9 reviewed both the white-box and the black-box techniques. He further identified, integrated and demonstrated a set of selected techniques for the information system professional. The test-case design techniques suitable for each level of computer-based testing along with the documents upon which they are based are listed in Table 2. The relationship between each level of computer-based testing and the documents with which it tests against is also depicted in Figure 2.

The Flow Of Computer-Based Testing Activities

Each level of computer-based testing consists of four major steps: (1) develop a test plan, (2) derive test cases and data, (3) execute the tests and (4) document and control the testing process. The first two stages are preparatory in nature but are of paramount importance. Without a well defined test plan and a set of derived test cases and data, computer-based testing of software will be like shooting without aiming at a target – of course, it will not be effective. The fourth stage (document and control the testing process) actually begins as soon as the first stage begins its course, and it continues even after the third stage ends. Many test-execution and documentation activities at one level of computer-based testing overlap concurrently with the preparatory test activities of the subsequent levels. Figure 3 shows the flow of computer-based testing activities that begins from the system requirements definition (SRD) phase and ends with the implementation (IMPL) phase. The development of Project Test Plan starts at the same time as the system requirements definition (SRD) phase and ends with the system design alternatives (SDA) phase. After the SDA phase, an individual test plan for each level of computer-based testing activities should be added to the Project Test Plan. By the end of the implementation (IMPL) phase, the Project Test Plan would contain a complete set of test plans developed throughout the entire project.

Note that Figure 3 does not show the manual testing activities such as walkthrough, reviews and inspections because these activities may be performed as frequently as needed throughout the entire system development life cycle. Another type of testing activity not shown in the figure is the regression testing. Regression testing is also a computer-based activity, yet it can be applied to any level of computer-based testing as frequently as necessary. It is usually performed after making a change to the program by rerunning the test cases that cover the program segment affected by the change.

Test Participants And Their Responsibilities

The participants of computer-based testing vary with the size of the system project. The larger the project, the larger the project team, and the more participants in the project. As suggested by Freedman and Weinberg,5 users—regardless of the project size-- should not be required to know technical details and thus should participate in only three levels of testing: system testing, software acceptance testing and final acceptance testing. Typically, in a small project with fewer than five people, the entire project team is responsible for all levels of testing except the two acceptance testing. For a medium or large project, the participants of each level of computer-based testing are somewhat different. The lowest level of testing should be conducted by each individual programmer. As the level elevates, individual programmers are left out to keep the size of the test team manageable.

There are five possible roles of testing paid by participants in computer-based testing. These five possible roles and their responsibilities are as follows:

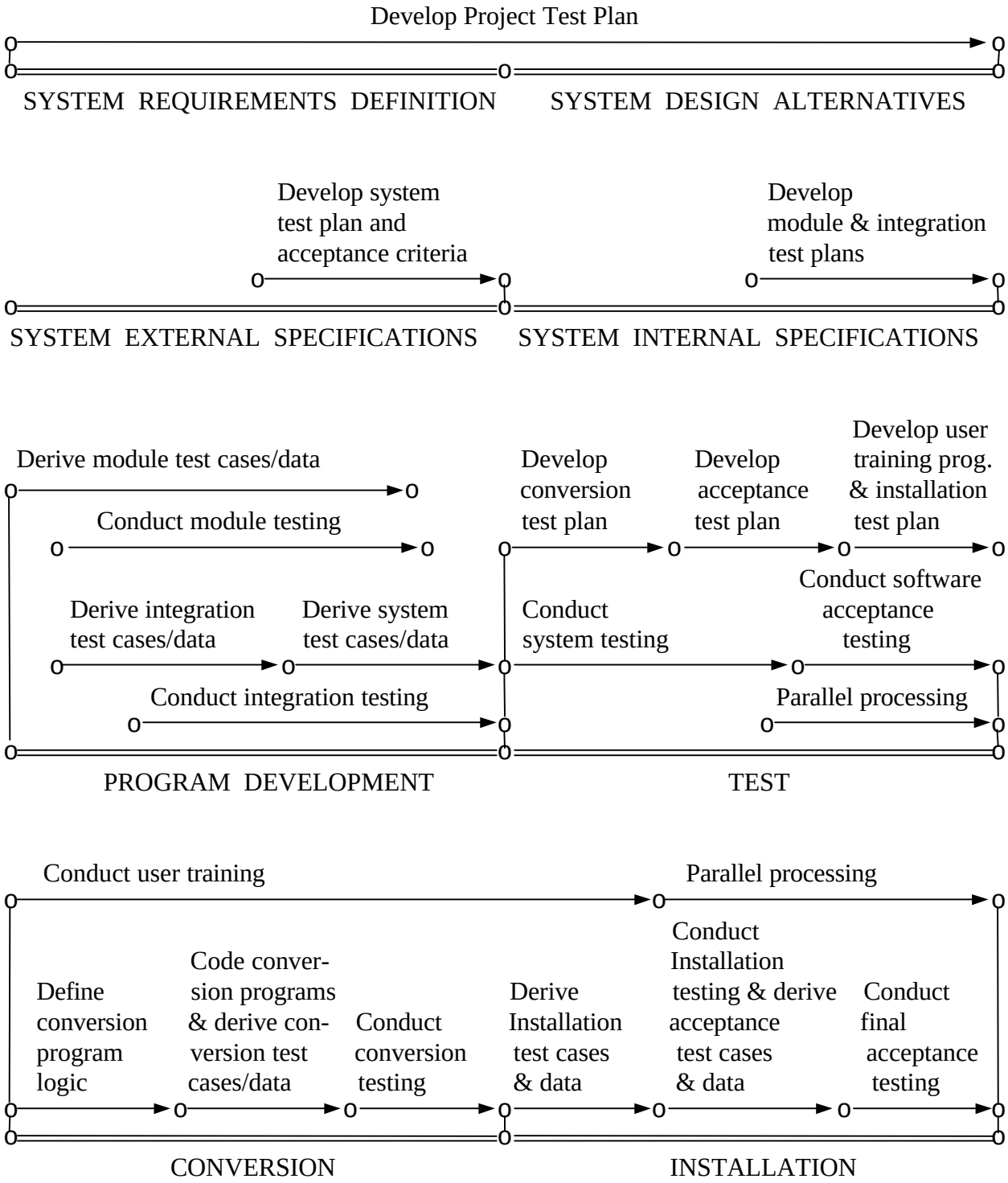

Figure 3
The Flow of Computer-Based Testing Activities

1. Test Data Administrator: Controls file structure and data base contents; maintains availability and recoverability of data; assists in developing and implementing the strategy of test data requirements.
2. **Test Executioner:** Prepares test-case specifications; creates or requests test data and files; performs desk checking and test run; prepares test results and discrepancy (error) report.
3. **Test Team Leader:** Participates in test planning; directs test preparation, execution and evaluation; creates test plan and test-case summary; requests testing supports; reviews the activities of test team; provides technical assistance to test team; attends quality assurance reviews and inspections.
4. Test Team Member: Participates in defining test conditions for test-case designs and reviewing test-case specifications and test results.
5. Test Supervisor/Coordinator: Assigns tests to test teams; reviews and approves all the relevant test materials.

LEVEL OF TESTING:	Module Tests	Program/Subsystem Integration Tests	System Integration Tests	System Tests	Software Acceptance Test	Conversion Tests	Installation Tests	Acceptance Test
MAJOR PARTICIPANTS:								
Individual Programmer	E	E				E		
Peer Programmers		E						
Programming Team Leaders	L	L	E	E	M	E		
Programming Team Supervisors	S	S	L	L	M	L	L	M
System Analysts		M	M	M	M			M
Project Manager			S	S	L	S	S	L
Quality Assurance (QA) Representatives	M	M	M	M	M	M	M	M
Test Data Administrator	D	D	D	D	D	D	D	D
Test Specialists			E	E	L,E	E	E	L,E
Users/User's Representatives				M	S,E			S,E

LEGEND: Role of Testing:
D = Test Data Administrator
E = Test Executioner
L = Test Team Leader
M = Test Team Member
S = Test Supervisor/Coordinator

Figure 4
Major Participants in Each Level of Computer-Based Testing

The possible participants in the systems project and their roles of testing in each level of computer-based testing are shown in Figure 4. Typically, in a medium or large project, each participant would play only one role except the test specialist and the user. Yet, in a small project, any participant may play more than one role. It is very likely that an IS professional would play each and every one of these roles throughout his entire IS career. Therefore, Figure 4 may serve as a very useful guideline to role playing for IS personnel.

Conclusion

There are various testing activities that a systems development project could entail. These testing activities are vital to each and every systems project since their application can dramatically reduce the risk of letting system errors go undetected and, thus, ensure the quality and on-time delivery of a system. Typically, such testing activities, along with the debugging activities, may take up 30-50% of the total project effort.3 '6 The actual allocation depends on the risk level of the application being developed. A higher testing and debugging effort

should be allocated to the application development that involved corporate-wide data interfaces rather than those that involved only a single functional area.

Furthermore, although the SDLC testing process and activities described in this article are in- tended for a Large and structured systems project, they can be tailored to a project of any size and structure, using any development approach. For example, a small project without a hardware installation may perform only manual testing (i.e., walkthrough, review, inspection, etc.), regression testing, module testing, program integration testing, system testing and final acceptance testing--skipping all other computer-based tests. Similarly, a small prototyping project may only carry out manual testing, regression testing, module testing, program integration testing and final acceptance testing. Moreover, if a project is large and the users' requirements are fuzzy, one may start with a prototyping process to develop the
system requirements definition (SRD). Once the SRD document is in place, the rigorous SDLC development and testing process can then be applied, if so desired.

One final note is that regardless of the test processes and activities, software testing can only help ensure the quality of a software product, rather than build it. To build a quality software product, one must proceed with a careful requirements analysis followed by an effective system design--rather than purposely rely on software testing and debugging to catch and re-move the errors that originate from the analysis and design process. ●jsm

References

1. Ascoly, J., M. Cafferty, S. Gruen, and O. Kohli, "Code Inspection Specification," TR-21 620, Kingston, New York: IBM System Communication Division, 1976.
2. Boar, B. H., *Application Prototyping*: *A Requirements Definition Strategy for the 80s*, New York: Wiley-lnterscience, 1984.
3. Boehm, B. W., "The High Cost of Software," *In Practical Strategies for Developing Large Software Systems*, edited by E. Horowitz, 3-14, Reading, MA: Addison-Wesley, 1975.
4. Boehm, B. W., "A Spiral Model of Software Development and Enhancement," *Computer*, 21,5 (May 1988): 61-72.
5. Freedman, D. P., and G. M. Weinberg, *Handbook of Walkthroughs, Inspects, and Technical Reviews*, Boston, MA: Little, Brown, 1982.
6. Davis, G. B., and M. H. Olson, *Management Information Systems: Conceptual Foundations, Structure, and Development*, New York: McGraw-Hill, 1985.
7. Lantz, K. E., *The Prototyping Methodology*, Englewood Cliffs, NJ: Prentice-Hall, 1986.
8. Larson, R. R., "Test Plan and Test Case Inspection Specifications," TR-21.586, Kingston, NY: IBM System Development Division, 1975.
9. Li, E. Y., "Software Testing Techniques for the Information Systems Professional: A Curriculum Perspective," *Proceedings of the 9th International Conference on Information Systems (1988)*: 1 19-128.
10. McCabe, T. J., and G. G. Schulmeyer, "System Testing Aided by Structured Analysis: A Practical Experience," IEEE *Transactions on Software Engineering*, SE-11, 9 (1985): 917-921.
11. Myers, G. J., *Composited/ Structured Design*, New York: Van Nostrand Reinhold, 1976.
12. Myers, G. J., *The Art of Software Testing*, New York: Wiley-lnterscience, 1979.
13. Naumann, J. D., and A. M. Jenkins, "Prototyping: The New Paradigm for Systems Development," *MIS Quarterly*, 6, 3 (September 1982): 29-44.
14. Perry, W. E., *A Structured Approach to Systems Testing*, Wellesley, MA: QED Information Sciences, 1983.
15. Waldstein, N. S., "The Walk-Thru: A Method of Specification, Design and Code Review," TR-00-2536, Poughkeepsie, NY: IBM System Development Division, 1974.
16. Wolverton, R. W., "The Cost of Developing Large-Scale Software," *IEEE Transactions on Computers* C-23, 6 (June1974): 615-636.