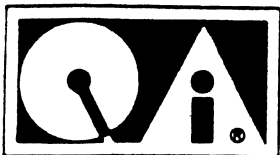


Quality DATA PROCESSING

Journal of the Quality Assurance Institute



July 1989



EDITORIAL BOARD

BOARD CHAIRMAN:

Wayne Smith, CQA, Applied Information Development, Inc.

BOARD MEMBERS:

James C. Badger, CQA, GTE Directories Service

Phillip N. Bentley, CQA, Kickstart Quality Systems

Ed Dreher, CQA, Pizza Hut, Inc.

Shirley Gordon, CQA, Consultant

Charles Hollocker, CQA, Northern Telecom, Inc.

John W. Horsch, CQA, Horsch & Associates

Peggy Myles, CQA, Contel Credit Corp.

Nancie L. Sill, CQA, Contel Credit Corp.

Linda T. Taylor, CQA, Taylor & Zeno Systems, Inc.

Gary C. Vennes, CQA, Norwest Technical Services

William E. Perry, CQA, Quality Assurance Institute, Managing Editor

Donna Baum, Production Editor

Martha Platt, Quality Assurance Institute, Assistant Editor

COLUMN EDITORS

Auditing—Keagle W. Davis, CPA, QAI Audit Division
Doing it Right the First Time—Jerome B.

Landsbaum, CQA, Monsanto Company

Education and Chapter News—William E. Perry, CQA, Quality Assurance Institute

The Lighter Side of Quality Assurance—William H.

Walraven, CQA, United Telecommunications

Quality Assurance Case Studies—Shirley Gordon, CQA, Consultant

Quality Assurance Surveys—William E. Perry, CQA, Quality Assurance Institute

Quality at Work—Rebecca Staton-Reinstein, CQA, New York Life Insurance Co.

Questions and Answers—Phillip N. Bentley, CQA, Kickstart Quality Systems

Speaking Out on Quality—Irv Brownstein, Consultant

Standards—Kenneth J. Muth, Target Stores

Testing—Harry Kalmbach, CQA, McDonnell Douglas Aerospace Information Services Co.

REGULAR CONTRIBUTORS TO *QUALITY DATA PROCESSING*

Robert Coull, CQA, GTE Data Services, Inc.

Mark Duncan, Federal Reserve Bank, Dallas

Victor M. Guarnera, CQA, Blue Cross/Blue Shield

Jerome B. Landsbaum, CQA, Monsanto Company

Daniel W. Martin, Tultex Corporation

Sara Messina, Manufacturers Hanover Trust

Leonard Muhs, CQA, New York State Department of Motor Vehicles

Robert E. Nichols, Federal Home Loan Mortgage Corporation

Ron Rizza, CQA, Ford Aerospace

W. Charles Slaven, CQA, General Electric Company

Robert J. Walsh, CQA, Putnam Investment Services

Donald E. Willett, RCA Automated Systems Division

Richard Zultner, CQA, Zultner and Company

TABLE OF CONTENTS

QDP JOURNAL AWARDS GLAVIN, SCHERKENBACH ARTICLES

Quality DATA PROCESSING again names winners of its editorial achievement awards in appreciation of the many distinguished contributions to the journal

TAKING QUALITY SERIOUSLY

Quality DATA PROCESSING shares with its readers comments by Colby H. Chandler, chairman and CEO at Eastman Kodak Company, on his company's pervasive effort to improve quality and cut costs

TESTING

Harry Kalmbach, CQA, Editor
McDonnell Douglas Aerospace Information Services Co.

"Structured Testing In The System Development Life Cycle"

Eldon Y. Li
Associate Professor and Coordinator
Management Information Systems
California Polytechnic State University

Defines structures testing, discusses its incorporation into the System Development Life Cycle, noting that the importance of structured testing process can never be overemphasized

QAI FEDERATION OF QUALITY ASSURANCE ASSOCIATIONS

A list of quality assurance associations currently in operation, or being formed, for members wanting to contact them

INDEPENDENT TEST TEAMS

Captain Rick Craig
U.S. Marine Corps

Offers independent test teams as a way to combat the problem of companies fielding new software products with countless undetected defects

CHARACTERIZATION OF SOFTWARE FOR TESTING

David M. Marks
Distinguished Member of Professional Staff
Bell Communications Research, Inc.

Presents the concept that software testing can be based upon a small set of functions which the software performs. The author discusses the determination of which software functions comprise a complete set, and proposes seven software functions

6

8

11

20

21

22

Harry Kalmbach, CQA, Editor
McDonnell Douglas Aerospace Information Services Co.

STRUCTURED TESTING IN THE SYSTEM DEVELOPMENT LIFE CYCLE

Eldon Y. Li
Associate Professor and Coordinator of
Management Information Systems
California Polytechnic State University

WHAT IS STRUCTURED TESTING?

The term "structured testing" has been defined by various authors. T. G. Lewis [18, p. 23] defines it as the process of top-down certification of software carried out by the software development team while the software is being developed. T. J. McCabe [20, p. iii] defines it as a process that: 1) measures and limits the complexity of program modules so they are more easily testable, 2) gives a quantification for how much testing should be done, and 3) contains a step-by-step procedure for obtaining the test data.

W. E. Perry [25, pp. 29-30] further advocates that application testing should be performed parallel to the system development life cycle (SDLC). In other words, when the system project starts, both the system development process and the application testing process begin at the same time. This practice can aid in early detection of requirements or design defects and allow timely corrections without ripple effects on the subsequent activities. Therefore, structured testing includes all validation, verification, and certification [15] activities. It is a step-by-step procedure for assuring the quality of test plans, designs, execution, controls, and documentation throughout the entire system development process.

Software testing usually goes hand-in-hand with software debugging. One must not confuse software testing with software debugging. The objective of software 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 (a deficiency), but if it does what it is not supposed to do (a malfunction). In contrast, the objective of debugging is to identify the type and the location of the "found" error and subsequently remove it by a redefinition, redesign, or recode, depending on the level of testing through which

the error was found. Thus, the former is destructive in nature while the latter is constructive, and they should be performed one right after another. In this paper, we shall focus our discussion on software testing and briefly describe the structured testing process in the context of a system development life cycle.

THE SYSTEM DEVELOPMENT LIFE CYCLE

The concept of the system development life cycle (SDLC) is similar to that of the product development life cycle in the manufacturing industry. The latter life cycle revolves around the six major functions: requirements, design, prototype, test, release, and follow-up. To be more specific, the process starts from analyzing consumers' requirements to determine product specification. Then, two or three product designs are developed. Product prototypes are built according to these designs. If no design can possibly meet the product specification or no prototyping can possibly be built to meet the product designs, the process would feed back and the product specification or designs should be modified.

Once the prototypes are constructed, they are tested. If a test failed, it may be due to errors in either the prototype building process or the product design, or even the product specification. Consequently, errors need to be corrected and the prototype needs to be rebuilt. If the prototypes passed all the tests, the best prototype will be selected and the final product design will be specified. If the selected prototype needs to be modified before it reaches the optimal conditions, it should be rebuilt and retested. If no changes are needed, the product is released and a new production process (developed in conjunction with the product design process) is implemented.

Continued

TESTING *Continued*

mented to manufacture this product. While manufacturing the product, a sales promotion is launched. After sales, the users' satisfaction toward the final product will be closely monitored.

Following a similar concept, the SDLC should revolve around six states: requirements, design, programming, test, implementation, and maintenance. In practice, these six stages may be further subdivided into various phases. R.G. Murdick (1970) had an extensive survey on the various approaches to such subdivision suggested in the literature. However, most of the proposed SDLC processes are "cascading" in nature. That is, they require that the current project phase be completed and/or signed off before the next phase is begun. This requirement was very popular during the early years because at that time

most applications were repetitive and well structured, and the users' requirements were relatively unchanged and could be clearly and accurately specified. However, in today's ever-changing competitive environment, computer applications are not limited to the repetitive and well-structured activities.

Users' requirements today are likely to be "fuzzy" (cannot be well defined) or "volatile" (constantly changing) requirements. Such types of requirements are very likely to cause severe errors at the requirements stage and fail a system project that adopts the SDLC process. This may be why 60 to 80 percent of all system errors were found to originate in users' requirements definition [6, pp. 17-18]. Such a high potential of misspecifying users' requirements anxiously calls for more flexible

FIGURE 1: THE OBJECTIVES OF THE SDLC PHASES

SDLC PHASES	PHASE OBJECTIVES	SDLC STAGES
Service Request/ Project Viability Assessment	To initiate a project and conduct cost/benefit analysis as well as a feasibility study.	Requirements
System Requirements Definition	To define project scope, analyze the existing system, and define information requirements, data attributes, and system objectives.	
System Design Alternatives	To identify and evaluate alternate system designs and prepare initial project schedules.	Design
System External Specifications	To specify data flow, user/system interface, system controls, and manual supporting procedures.	
System Internal Specifications	To specify processing logic, file structure, module interfaces, and system architecture.	
Program Development	To transform programs' internal specifications into program code using a computer language.	Programming
Testing	To verify and validate the system being developed throughout the SDLC.	Test
Conversion	To convert the data formats and procedures for the new system.	Implementation
Implementation	To install the hardware and software for the new system, and cut over the system into production.	
Postimplementation Review/Maintenance	To monitor and maintain the quality and performance of the new system.	Maintenance

Continued

alternate approaches such as "iterative" [4, 11], "heuristic" [5], "evolutionary" [12,13], "accelerated" [13], and "prototype" [3, 6, 10, 16, 24] development processes. All these alternate approaches allow feedback to the early stages in the SDLC process, especially the requirements definition. Without such feedback, the final product is likely to require a major overhaul; otherwise, it is nothing but a "throwaway" [9].

Recent advancement in fourth generation languages,

"AFTER SALES, THE USERS' SATISFACTION TOWARD THE FINAL PRODUCT WILL BE CLOSELY MONITORED."

computer-aided software engineering tools, automated code generators, and automated document management systems, among others, allows easy modifications to the system in progress and thus makes such feedbacks not only possible but also real. It is this feedback loop which closes the gap between the end user and the system builder, and makes the SDLC process resemble the product development process.

To facilitate our discussion, we shall subdivide the six SDLC stages into ten phases. These ten phases follow the SDM/70 methodology which was developed by Atlantic Software, Inc. in conjunction with Katch & Associates. Figure 1 shows the objectives of these ten phases within the six-stage SDLC framework. Depending on the project schedule and the characteristics of users' requirements, these ten phases may be tailored to a particular application development project. For example, a project with a short time frame could replace the system external specifications, system internal specifications, and program development phases with a software acquisition phase or, alternatively, replace the system requirements definition, system design alternatives, system external specifications, system internal specifications, and program development phases with a prototyping process. Likewise, a project with "fuzzy" users' requirements could use an iterative prototyping process to develop its system requirements definition [6, 16] while a project with "volatile" requirements could use a similar prototyping process to develop a working, operational system [3, 10, 24]. In this paper, we shall discuss the structured testing process using the full ten phases of the SDLC.

STRUCTURED TESTING IN THE SYSTEM DEVELOPMENT LIFE CYCLE

As alluded earlier, structured testing should start as soon as the project is launched (i.e., at the beginning of the SDLC process). At the first five phases of the SDLC (before any module coding is completed), manual testing techniques such as structured walk throughs [27], desk checking, reviews [14], and inspections [2, 17] are used to verify that the end products of the current SDLC phase are correct based upon the output of the prior phases. For

example, at the end of the system requirement definition (SRD) phase, the SRD document should be verified by comparing it to the service request and users' current opinions. If any mistakes or necessary changes have been discovered, they will be fed back to the SRD process. Otherwise, the SRD documents will be approved or signed off by the user and the system design alternatives phase is then begun. Such manual testing can be applied to any project document throughout the entire life cycle.

Once the program development phase is begun, computer-based testing will soon come into play in addition to manual testing. Computer-based testing requires the program code to be compiled and executed by a computer; therefore, it cannot be performed until at least one program module has been completely coded. According to the testing objectives, computer-based testing can be classified into seven levels: 1) module (unit) testing, 2) integration (interface) testing, 3) system testing, 4) software acceptance testing, 5) conversion testing, 6) installation testing, and 7) final acceptance testing, each of which focuses on a particular class of errors. These seven levels of testing and their corresponding test objectives are exhibited in Table 1.

"WITHOUT SUCH FEEDBACK, THE FINAL PRODUCT IS LIKELY TO REQUIRE A MAJOR OVERHAUL; OTHERWISE, IT IS NOTHING BUT A 'THROWAWAY'."

THE FUNDAMENTAL STRUCTURED TESTING PROCESS

Each and every level of computer-based testing entails a four-stage fundamental structured testing process. The stages and the purposes of this process are discussed in sequence.

1. **Develop a test plan.** To provide a guideline for the project team to plan for the test sequence/phases, test criteria, test schedule, and the required supporting resources such as personnel, hardware, software, test data, test techniques, test tools, and budget support.
2. **Derive test cases (or test scripts) and test data.** To derive a set of test cases (or test scripts) that cover as many test conditions or test paths in the program or system as possible, and to derive the input data and their corresponding expected output for each test case or script.
3. **Execute the tests.** To compile and execute the intended program or system on a computer using the derived test data as the input, and to compare the resulted output with the expected output specified in Step 2.
4. **Document and control the testing process.** To document and summarize the test plan, the test cases/scripts, the test data, and the test results in

Continued

TABLE 1: THE LEVELS OF COMPUTER-BASED TESTING AND THEIR CORRESPONDING OBJECTIVES.

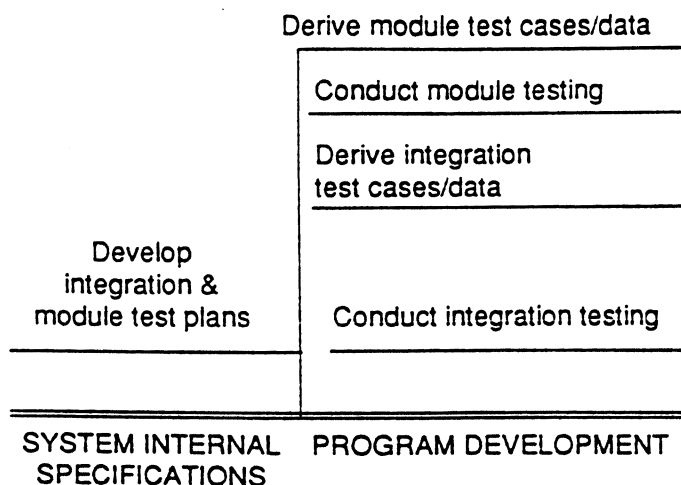
LEVEL OF TESTING	OBJECTIVES OF TESTING
Module (Unit)	To test a module (i.e., a subprogram, a subroutine, or a Testing procedure) in a program to see if the module contradicts the system internal specs.
Integration (Interface) Testing	To merge and test program modules to see if they can work correctly as a whole without contradicting the system's internal and external specifications.
System Testing	To verify that the system as a whole is structurally and functionally sound and meets the system requirements definition, the system design alternatives, and the system external specs.
Software Acceptance Testing	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.
Conversion Testing	To test the file conversion program to see that its program logic meets the specification, that the program contains all necessary control procedures, and that the conversion process yields the expected results.*
Installation Testing	To ensure that: 1) a 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.*
Final Acceptance Testing	To ensure that the system meets its initial requirements, system objectives, test criteria, and the current needs of its end users.*

* Adapted from Myers [23].

order to provide sufficient information for project management and control, and to monitor and communicate any changes in the test plan, test cases/scripts, test data, and test resources. The activities in this step actually intersperse throughout the entire structured testing procedure.

Note that the first two stages are preparatory in nature but are of paramount importance. Without a well-defined

FIGURE 2: THE STRUCTURED PROCESSES FOR MODULE TESTING AND INTEGRATION TESTING

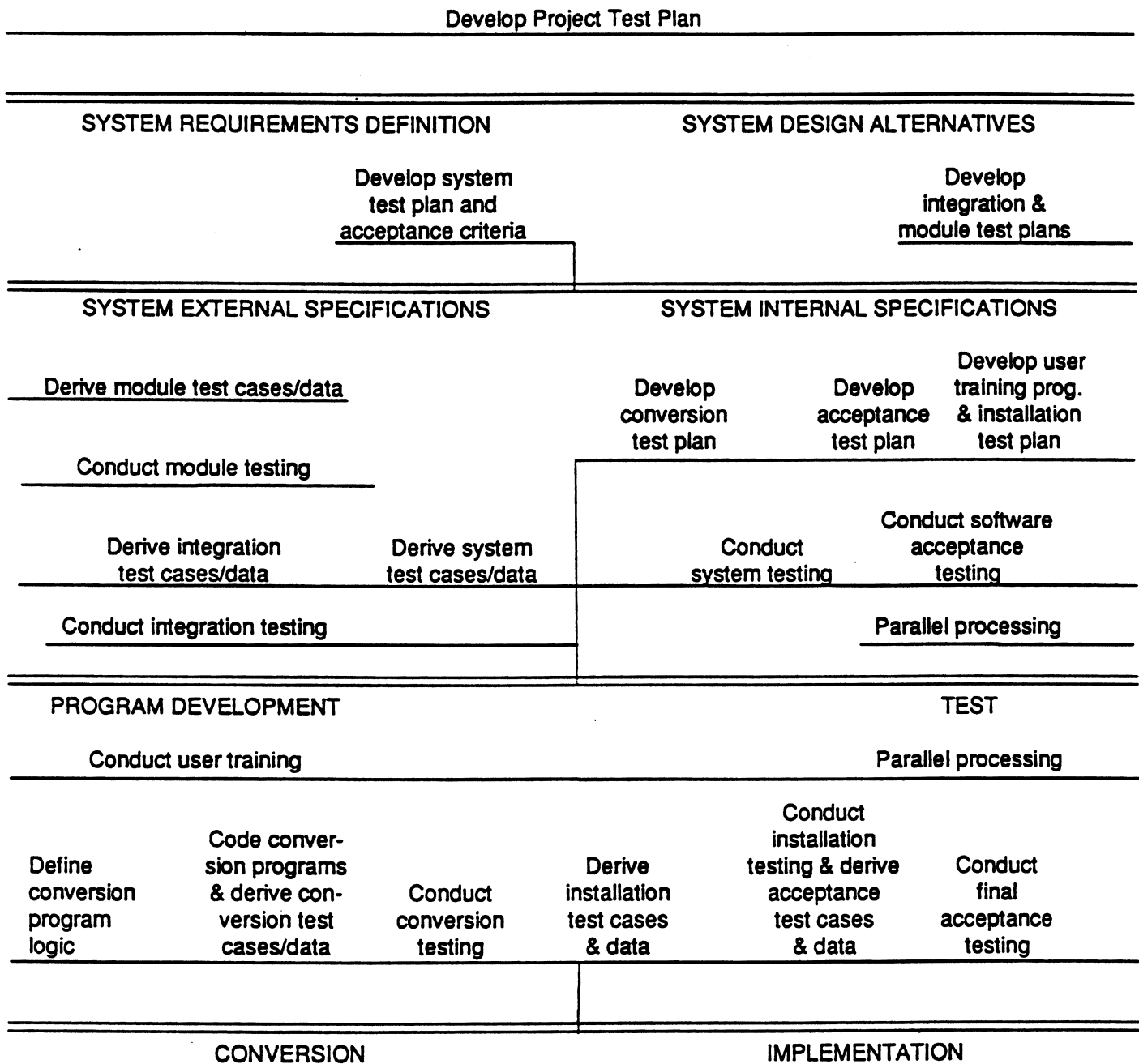


test plan and a set of carefully derived test cases and data, computer-based testing will not be effective.

As an example, Figure 2 depicts the structured processes for module testing and integration testing. The test plans of these two levels of testing should be developed at the same time, long before one starts to derive test cases and test data (usually in the system internal specifications phase in which the programs' internal specifications are completed). The module test plan should take into account the integration strategy (either top-down or bottom-up, or a mixture of both) laid out in the integration test plan because the strategy may require that some modules be tested before others. Once both test plans are in place, the test cases and test data for an intended program module are then derived from its internal specification. While the test cases and test data are being derived, the module is being coded. Once the module coding is completed, it is tested on a computer with the derived test data as the input. The test results are subsequently documented. In most cases, the testing process for one program module overlaps those for the others. That is, when a module test is being executed, other programmers might be testing or deriving the test cases and test data for the other program modules. By the same token, the structured testing steps for integration testing usually overlap. Nevertheless, the test exe-

Continued

FIGURE 3: A STRUCTURED PROCESS FOR COMPUTER-BASED TESTING IN THE SYSTEM DEVELOPMENT LIFE CYCLE



cution must wait until the modules to be integrated are all independently tested. Both the module testing and the integration testing are carried out within the program development phase, therefore, they are also known as the "development testing" [28].

Similarly, many test execution and documentation (steps 3 and 4) activities at one level of computer-based testing may overlap the test planning and test case design (steps 1 and 2) activities of the subsequent levels. Figure 3 shows a Gantt chart depicting a possible structured process for all levels of computer-based testing in the context of the system development life cycle. This

figure provides a valuable road map of computer based structured testing in a system project.

STRUCTURED TECHNIQUES FOR DERIVING TEST CASES

There are several structured techniques for deriving test cases at each level of computer-based testing. These techniques may be classified into two categories: the white-box techniques, and the black-box techniques [23]. White-box techniques (also called structural, code-based, or logic-driven techniques) require the tester to

Continued

TESTING *Continued*

examine the internal structure of the program and derive the test cases and test data from the program logic described in the system internal specifications 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 test data are derived solely from the system requirements definition or the system external specifications. The existing white-box techniques include: 1) statement coverage, 2) decision coverage, 3) condition coverage, 4) decision/condition coverage, 5) multiple condition coverage, and 6) complexity-based coverage. The black-box techniques include: 1) equivalence partitioning, 2) design-based equivalence partitioning, 3) cause-effect graphing, 4) boundary value analysis, and 5) error guessing. A review of each of these techniques can be found in Adrion, et al. [1], Li [19], and Myers [23].

THE ROLES OF TEST PARTICIPANTS

The participants of computer-based testing usually 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 [14], the user, regardless of 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 test. Typically, in a small project with a size of less than five persons, the entire project team is responsible for all levels of testing except the two acceptance testings. 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 of sight so as to keep the size of the test team manageable. All participants in a com-

FIGURE 4: MAJOR PARTICIPANTS IN EACH LEVEL OF COMPUTER-BASED TESTING

LEVEL OF TESTING:	Module Tests							
	Program/Subsystem Integration Tests							
	System Integration Test							
	System Test							
	Software Acceptance Test							
	Conversion Tests							
MAJOR PARTICIPANTS:	Installation Tests							Acceptance Test
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
Systems Analysts		M	M	M	M			M
Project Manager		S	S	L	S	S	L	
Quality Assurance (QA)	M	M	M	M	M	M	M	M
Representatives								
Test Data Administrator	D	D	D	D	D	D	D	D
Test Specialists		E	E	L,E	E	E	L,E	
Users/Users' 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

Continued

TABLE 2: RECOMMENDED ALLOCATION OF TESTING AND DEBUGGING EFFORT IN THE SDLC

SDLC STAGES	ESTIMATED PERCENT OF PROJECT EFFORT*	PROJECT EFFORT ALLOCATED TO TESTING & DEBUGGING	PERCENT OF ALLOCATED TESTING DEBUGGING EFFORT	PERTINENT TEST ACTIVITIES**
Requirements	15%	3.0%	10%	Manual tests
Design	30%	9.0%	30%	Manual tests
Programming	15%	7.5%	25%	Manual tests, Module tests, and Integration tests
Test	30%***	6.0%	20%	Manual tests, System test, and Software acceptance test
Implementation	10%	4.5%	15%	Manual tests, Conversion tests, Installation tests, and Final acceptance test
Total Effort	100%	30.0%***	100%	

* Documentation effort is included in the estimates.

** Manual tests include walkthroughs, reviews, and inspections.

*** This estimate includes all manual and computer-based testing and debugging activities throughout the system development life cycle. It also includes test planning, designs, executions, controls, and documentation.

puter-based testing can be classified into five different roles. These five roles and their responsibilities are:

- 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.

The possible participants in a system 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.

ALLOCATION OF TESTING AND DEBUGGING EFFORT

A system project may spend between 30 to 50 percent of its total development effort in testing and debugging [7, 23, 28]. The actual allocation may depend on the risk level of an application being developed. One plausible way of determining the risk level of an application is to assess the size, the structure, and the technology that application entails [21, 26]. A higher degree of testing and debugging effort should be allocated to an application

Continued

that involved high risk than one that involved low risk. In general, the more complex the application, the higher the percentage of the development effort that should be allocated to testing and debugging.

Table 2 shows the recommended allocation of the testing and debugging effort for the first five stages of the system development life cycle. This recommendation was based on the concept that software quality must be built in, not added on, and that the quality building process should start from careful requirements analysis followed by effective system design and programming practice. As indicated in Table 2, the percent of project effort allocated to the first three stages may amount to 60 percent. If effort is effectively spent on these three stages, one can expect that less testing and debugging effort will be needed. Therefore, only 30 percent of the project effort is recommended for all testing and debugging activities. This allocation includes all the test-related activities, namely, test planning, designs, executions, controls, and documentation.

Traditionally, the effort allocated to the manual testing activities such as walk throughs, reviews, and inspections was almost nil. Ironically, these manual activities are the only suitable testing activities for the first two SDLC stages (i.e., requirements and design). A reasonable proportion (about 40 percent) of the allocated test effort should be directed to the walk throughs, reviews, and inspections of system requirements and designs. As indicated in Table 2, about 15 percent of the total project effort should be allocated to requirements analysis and definition, and 30 percent to system design. It is recommended that 10 percent of the total testing and debugging effort (or 3 percent of the total development effort) be assigned to the requirements stage, and 30 percent (or 9 percent of the total development effort) to the design stage.

Recently, there is a trend of using a code generator to replace human programmers for the program coding activities. Such practice may very well reduce human errors in translating the design specifications into computer language code. Yet, it does not reduce any design error which might arise in the earlier phases. Often, many design errors can only be detected by testing the program code on a computer. Therefore, the use of a code generator does not exempt the program code from being subjected to computer-based testing. However, one can expect that the testing and debugging effort allocated to the programming stage will be dramatically reduced since no human errors will occur in the program coding process that will need to be detected and removed.

SUMMARY

The importance of the structured testing process can never be overemphasized. First, it advocates that an application testing process should start as soon as the development process begins. This practice can bring

forth early detection of errors in requirements to avoid "ripple" effects on the subsequent project phases. Second, it provides a set of well-defined steps to control software quality. By following these steps, one can dramatically reduce the risk of letting system errors go undetected and thus ensure the quality and on-time delivery of a system development project. Third, the structured testing process demands high standards in the controls and documentation of a system project. Such standards may, in effect, result in a reduction in future maintenance cost.

Although the structured testing process is most suitable to a large and complex system project having well-defined requirements, it can also be applied to a project with "fuzzy" (unclear) or "volatile" (changing) users' requirements. The development of the latter types of projects requires that a prototyping process be incorporated into the SDLC process. Yet, to test a prototype, one can still apply the structured testing process.

To close our discussion, we have compiled below a list of test principles based on the existing literature [1, 8, 22, 23] and the author's experience.

Principles of Structured Testing


- Plan a testing effort under the assumption that some errors will be found.
- A programming group should not test their own programs alone. They should be accompanied by an independent agent such as a quality assurance (QA) representative and/or test specialist.
- One who performs software coding should test his/her own code before someone else does it.
- Generate test data at all stages and, in particular, the early stages.
- Inspect requirements, design, and code for errors and for consistency.
- Be systematic in your approach to testing. Standards, guidelines, and procedures should be established.
- Test cases must be written for invalid and unexpected, as well as valid and expected, input/output conditions.
- A necessary part of every test case is a description of the expected output or results.
- Each program must have an error-free compilation before it is formally tested.
- Avoid nonreproducible or on-the-fly testing. Document your test sequence and input stream.
- Test pieces and then aggregates.
- Save, organize, and annotate test runs.
- Thoroughly inspect the results of each test.
- The probability of the existence of more errors in a section of a program is proportional to the number of errors already found in that section. Thus, one should concentrate testing on modules that exhibit the most

Continued

errors and on their interfaces.

- Retest (through regression testing) when modifications are made.
- Discover and use available tools on your system.
- Never alter the program to make testing easier.
- Ensure that testability is a key objective in your software design.
- The design of a system should be such that each module is integrated into the system only once.
- Testing is an extremely creative and intellectually challenging task. Thus, one should assign his/her most creative programmers to do the tests.

REFERENCES

1. Adrion, W. R., Branstad, M. A., and Cherniavsky, J. C. "Validation, Verification, and Testing of Computer Software," *ACM Computing Surveys*, Vol. 14, No. 2 (June 1982), pp. 159-192.
2. Ascoly, J., Cafferty, M., Gruen, S., and Kohli, O. "Code Inspection Specification," TR-21.630, Kingston, NY, IBM System Communications Division, 1976.
3. Bally, L., Brittan, J., and Wagner, K. H. "A Prototype Approach to Information Systems Design and Development," *Information and Management*, Vol. 1, No. 1 (November 1977), pp. 21-26.
4. Basili, V. R., and Turner, A. H. "Iterative Enhancement: A Practical Technique for Software Development," *IEEE Tutorial: Structured Programming*, #75ch1049-6, Sept. 1975.
5. Bernisford, T. R., and Wetherbe, J. C. "Heuristic Development: A Redesign of Systems Design," *MIS Quarterly*, Vol. 3, No. 1 (March 1979), pp. 11-19.
6. Boar, B. H. *Application Prototyping: A Requirements Definition Strategy for the 80s*, New York, Wiley-Interscience, 1984.
7. Boehm, B. W. "The High Cost of Software," In *Practical Strategies for Developing Large Software Systems*, E. Horowitz, editor, Reading, MA, Addison-Wesley, 1975, pp. 3-14.
8. Branstad, M. A., Cherniavsky, J. C., and Adrion, W. R. "Validation, Verification, and Testing for the Individual Programmer," *Computer*, Vol. 13, No. 12 (December 1980), pp. 24-30.
9. Brooks, F. *The Mythical Man-Month: Essays on Software Engineering*, Reading, MA, Addison-Wesley, 1975.
10. Canning, R. G. "Developing Systems by Prototyping," *EDP Analyzer*, Vol. 19, No. 9 (September 1981), pp. 1-14.
11. Davis, G. B. and Olson, M. H. *Management Information Systems: Conceptual Foundations, Structure, and Development*, New York, McGraw-Hill, 1985.
12. Edelman, F. "The Management of Information Resources: Challenge for American Business," *MIS Quarterly*, Vol. 5, No. 1 (March 1981), pp. 17-27.
13. Edelman, F. "Position Paper No. 2: Evolutionary Development—An Efficient Strategy for Systems Design," unpublished paper, October, 1981.
14. Freedman, D. P. and Weinberg, G. M. *Handbook of Walkthroughs, Inspections, and Technical Reviews*, Boston, MA, Little, Brown, 1982.
15. Jensen, R. W., and Tonies, C. C. *Software Engineering*, Englewood Cliffs, NJ, Prentice-Hall, 1979, pp. 553-567.
16. Lantz, K. E. *The Prototyping Methodology*, Englewood Cliffs, NJ, Prentice Hall, 1986.
17. Larson, R. R. "Test Plan and Test Case Inspection Specifications," TR 21.586, Kingston, NY, IBM System Development Division, 1975.
18. Lewis, T. J. *Software Engineering: Analysis and Verification*, Reston, VA, Reston Publishing, 1982.
19. Li, E. Y. "Software Testing Techniques for the Information Systems Professional: A Curriculum Perspective," *Proceeding of the 9th Annual International Conference on Information Systems (ICIS '88)*, Minneapolis, MN 1988.
20. McCabe, T. J., editor, *Structured Testing*, Silver Spring, MD, IEEE Computer Society Press, 1983.
21. McFarlan, F. W. "Portfolio Approach to Information Systems," *Harvard Business Review*, (September-October 1981), pp. 142-150.
22. Myers, G. J. *Software Reliability: Principles and Practices*, New York, Wiley-Interscience, 1976.
23. Myers, G. J. *The Art of Software Testing*, New York, Wiley-Interscience, 1979.
24. Naumann, J. D., and Jenkins, A. M. "Prototyping: The New Paradigm for Systems Development," *MIS Quarterly*, Vol. 6, No. 3 (September 1982), pp. 29-44.
25. Perry, W. E. *A Structured Approach to Systems Testing*, Wellesley, MA, QED Information Sciences, 1983.
26. Perry, W. E. *A Standard for Testing Application Software*, Boston, MA, Auerbach Publishers, 1987.
27. Waldstein, N. S. "The Walk-Thru: A Method of Specification, Design, and Code Review," TR-00-2536, Poughkeepsie, NY, IBM System Development Division, 1974.
28. Wolverton, R. W. "The Cost of Developing Large-Scale Software," *IEEE Transactions on Computers*, Vol. C23-No. 6 (June 1974), pp. 615-636. 

ELDON Y. LI

Eldon Y. Li is an Associate Professor and the Coordinator of Management Information Systems Program at the School of Business, California Polytechnic State University, San Luis Obispo. He received his Ph.D. from Texas Tech University. He has provided consulting services to many firms including Bechtel and IBM, and to the clientele of the U.S. Small Business Administration. His current research interests include systems analysis and design, software engineering, human factors in information systems, expert systems, and information management. He is a Certified Data Educator (CDE), a certified professional in production and inventory management (CPIM), and a member of IEEE-Computer, ACM, AAAI, DSI, and TIMS.

—QAI MOTTO—

*"The bitterness
of poor quality
remains long after
the sweetness of
meeting the schedule
has been forgotten."*

Best Paper
1989



Eldon
Y.
Li

Significant Contribution
1989



Alice
E.
Fugate

QUALITY DATA PROCESSING 1989 WRITING AWARD WINNERS EXEMPLIFY VARIED APPROACHES TO QUALITY

Articles with varied approaches to quality, one stressing the importance of structured testing and one dealing with the psychological impact of change, were named winners of the *Quality Data Processing* 1989 writing awards.

"Structured Testing in the System Development Life Cycle" (July 1989), by Eldon Y. Li, was named "Best Paper 1989," while "Managing Change: How to Sell Users on Innovation" (April 1989), by Alice E. Fugate, won the designation of "Significant Contribution 1989."

Li's article defines structured testing and discusses its incorporation into the systems development life cycle. It stresses the importance of the structured testing process.

Fugate's article takes a look at the psychological impact of change and what it means for those faced with managing the process. Fugate offers techniques managers can use to help staff members cope with technical innovation.

The awards were announced by *Quality Data Processing* Editorial Board Chairman Wayne Smith, CQA, at the 1990 International Conference on Information Systems Quality Assurance in Orlando, Florida, Thursday, April 26, 1990.

Li, an associate professor and former coordinator of management information programs at the School of Business, California Polytechnic State University, has a Ph.D. in business administration from Texas Tech. A former management consultant to clientele of the U.S. Small Business Administration and software quality consultant for Bechtel Corporation, as well as a visiting software scientist for IBM Corporation, Li's current research interests include planning and control of informa-

tion systems, software quality engineering, systems analysis and design, human factors in IS, expert systems, and information management.

Fugate is a self-employed writer and editor based in St. Louis. Her experience includes working on college text books and corporate communications in the areas of business, information systems, science, and the humanities. Formerly with The Center for the Study of Data Processing at Washington University, Fugate has also worked as a developmental editor for Mosby Publishing. She is a member of the steering committee for St. Louis Women's Commerce Association, a group that named her 1989 Woman of Recognition.

Presentation of the awards marked the third year that *Quality Data Processing*, the journal of the Quality Assurance Institute, gave awards to the outstanding contributors to the quarterly publication.

1988 winners were William F. Glavin, whose article, "Quality, the Path to Customer Satisfaction and Leadership" (July 1988), was named "Best Article 1988," and William W. Scherkenbach, whose article, "The Meaning of Competitiveness" (January 1988), was named "Significant Contribution 1988."

1987 winners were Patrick L. Townsend and Barbara K. Kimball, coauthors of "A Quality Compound" (October 1987), named "Best Article 1987," and Jerome B. Landsbaum, author of "Using Measurement to Improve Productivity" (January 1987), named "Significant Contribution 1987."

While naming the 1989 award winners, Smith cited several other articles published in 1989 that he considered important articles, important enough to be read again and to be useful in the workplace. Ten additional

Continued

articles considered to be serious candidates for awards were given the designation of "Honorable Mention."


Those articles, in order of date of publication are: "Measuring Software Defect Removal" (January 1989), by Capers Jones; "MIS Quality Survey" (January 1989), prepared by John Diebold and Associates and submitted by Fred Knotek; "Systems Excellence by Design" (April 1989), by Anthony W. Crawford; "Automated Change Control—Buy Versus Build" (April 1989), by Thor Hoff; "Achieving A Total Commitment to Quality Within the Software Development Environment" (April 1989), by Barbara Hirsh; "Characterization of Software for Testing" (July, 1989), by David M. Marks; "Testing for Usability" (July, 1989), by Kathleen M. Potosnak and Richard P. Koffler; "Customer Quality Assessment Visits" (October 1989), by Robert W. Shillato; "A Guide to Understanding the Requirements Definition Process" (October, 1989), by Beverly B. Madron; and "Do You Know if Your Colleagues Like Your Standards?" (October 1989), by Valerie J. Alt.

Articles were judged by the 1989 *Quality Data Processing* Editorial Board using the same criteria by which it selects articles for publication in the journal. Those criteria include: advancing the practice of quality assurance, representing a balance between management disciplines, organizational strategies, technical methodologies, and administrative procedures. In screening

articles, the board looks for those that are suitably diverse, offering fresh insights, and relevant to the management information systems situation, sensitive to its practicalities and constraints, and having a format, structure, and presentation that exemplify a high level of professionalism.

Editorial board members review all articles for publication in the journal. The 1990 board includes Smith, Shirley Gordon, CQA, Charles Hollocker, CQA, John Horch, CQA, Harry Kalmbach, CQA, Eldon Y. Li, Ph.D., DPIM, CDE, Peggy Myles, CQA, Nancie L. Sill, CQA, Rebecca Staton-Reinstein, CQA, and Linda T. Taylor, CQA.

Smith noted that the journal consistently receives articles of high quality as well as a high volume of articles. "The diversity of the 1989 winning articles reflects a principle that we promote in the journal, that of a need for cross-disciplinary skills and approaches. These articles exemplify the need for the quality assurance function to emphasize human factions as well as technical skills," he said.

"The journal is a most effective way for data processing practitioners to share quality techniques. We encourage people to submit their ideas, and will again recognize top contributions next year," said William E. Perry, editor of *Quality Data Processing* and executive director of the Quality Assurance Institute. 

QUALITY ASSURANCE SPECIALIST

Job Opening

Come to the Pacific Northwest!

Job Responsibilities:

- Standardizing our Systems Development Life Cycle, including Feasibility Studies, RFP's, Project Plans, etc.
- Policy Development.
- User satisfaction metrics.
- Request and time tracking systems.
- Documentation standards.
- Security Officer

Qualifications:

- Several years experience as a QA analyst.
- Preference will be given to analysts with a Certified Quality Assurance certificate.
- Preference given to candidates with mini-computer background in a hospital environment.

Salary to \$40K+

Legacy is a multi-hospital organization with over 9,000 employees.

Send resumes to :

Legacy Health System, Human Resources, 500 NE Multnomah, Portland OR, 97232

or call 503-225-4370 for more information.

An Equal Opportunity Employer