



# Plan Into Practice Software Testing at Tandem Computers

Keith Stobie  
Mary Alexander

Technical Report 87.3  
May 1987  
Part Number 83055



Plan Into Practice  
Software Testing at Tandem Computers

Keith Stobie  
Mary Alexander

May 1987

Tandem Technical Report 87.3



Plan Into Practice  
Software Testing at Tandem Computers

Keith Stobie  
Mary Alexander

May 1987

ABSTRACT

Tandem Computers is a manufacturer of OnLine Transaction Processing mini- and mainframe computer systems. Software quality is very important at Tandem, since a software failure can nullify any hardware fault tolerance and make resources unavailable. Tandem's software development organization does most of the product oriented testing. Tandem has approximately one quality assurance (QA) developer for every three product developers. Tandem follows a fairly typical software life cycle but has several innovations to allow ample time for testing and user feedback. Rather than adversaries, at Tandem product and QA developers are on the same team with a common goal, that of delivering high quality, reliable software. Tandem SQA bases several of its documents on the IEEE Standard for Software Test Documentation.

Several tools developed in-house and currently in use are described. Automated Library Environment is used to control and catalog the running of tests. Dynamic Reload controls the automatic dumping and reloads of a system. Terminal Simulator provides record and playback of Tandem supported terminals. JET is a PC based terminal simulator. COVER measures code coverage. Product Reporting System is used for tracking incidents.

Tandem and NonStop are trademarks of Tandem Computers Incorporated.



# Contents

1.0	Introduction	1
1.1	Organization	2
2.0	Process	4
2.1	QA Training	4
2.2	Philosophy	5
2.3	Product Requirements	6
2.4	QA Strategy Document	6
2.5	External Specifications	7
2.6	QA Test Plans	7
2.7	Internal Design	8
2.8	Product Development Test Plans	8
2.9	Code and Test	9
2.10	Test Design Specifications	9
2.10.1	Test Library	10
2.11	Test Cases	11
2.12	Developer Release	11
2.13	Release Build	12
2.14	Manuals	14
2.15	Classes	14
2.16	Systems QA	14
2.17	Alpha Testing	15
2.18	Beta Testing	16
2.19	Controlled Distribution	17
2.20	General Availability	17
3.0	Tools & Measurements	19
3.1	ALIEN	20
3.1.1	Example ALIEN reports	23
3.1.2	Weekly Release Testing Reports	25
3.1.3	Example cross product weekly summary	25
3.2	QACOMP	26
3.3	DIVER & DYNREL	27
3.4	TSIM	28
3.5	JET	29
3.6	Product Specific Testing Tools	30
3.7	VC	30
3.8	PRODRLSE	31
3.9	COVER	31
3.9.1	Example COVER summary output	33
3.10	PRS	34
3.10.1	Sample TPR	34
4.0	Conclusion	38
4.1	IEEE Standard for Software Test Documentation comparison	38
4.2	FUTURE DIRECTIONS	39
4.3	SUMMARY	40
4.4	Acknowledgements	40
REFERENCES		41





## 1.0 Introduction

Tandem Computers Incorporated manufactures and markets computer systems and networks for on-line transaction processing. Typical examples of on-line transaction processing are automatic teller machines, airline reservation systems, electronic funds transfer, and point-of-sale systems. The fault-tolerant NonStop systems range in size from mini to some of the world's largest mainframes. Tandem has over 6000 employees in 130 locations around the world.

Most software development at Tandem is done at corporate headquarters in Cupertino, California. There are approximately 500 software developers, with about one fourth of them being QA developers. Our present overall ratio of product to QA developers is 3.0 to 1. Tandem's software products comprise approximately ten million lines of code. Several major software releases a year recompile the majority of the code, and requires Tandem to test all software.

Management's strong commitment to Software Quality at Tandem is easy to understand, since a software failure can nullify hardware fault tolerance and make resources unavailable. Customers expect reliability from any vendor promoting high availability, and Tandem's continued success in the marketplace hinges on its ability to continue to meet these expectations.

This paper describes some of the approaches used at Tandem to test software, ensure its quality, and evaluate the results. The techniques described are divided into three broad areas: Processes, Tools, and Measurements.

## 1.1 Organization

Tandem's Software Development organization maximizes our ability to produce high-quality software in the shortest possible time. Our focus is on building the quality in, not on testing the bugs out. Our own experiences match testing theory; it is cheaper to keep defects out of the software in the first place than to find and remove them later in the cycle.

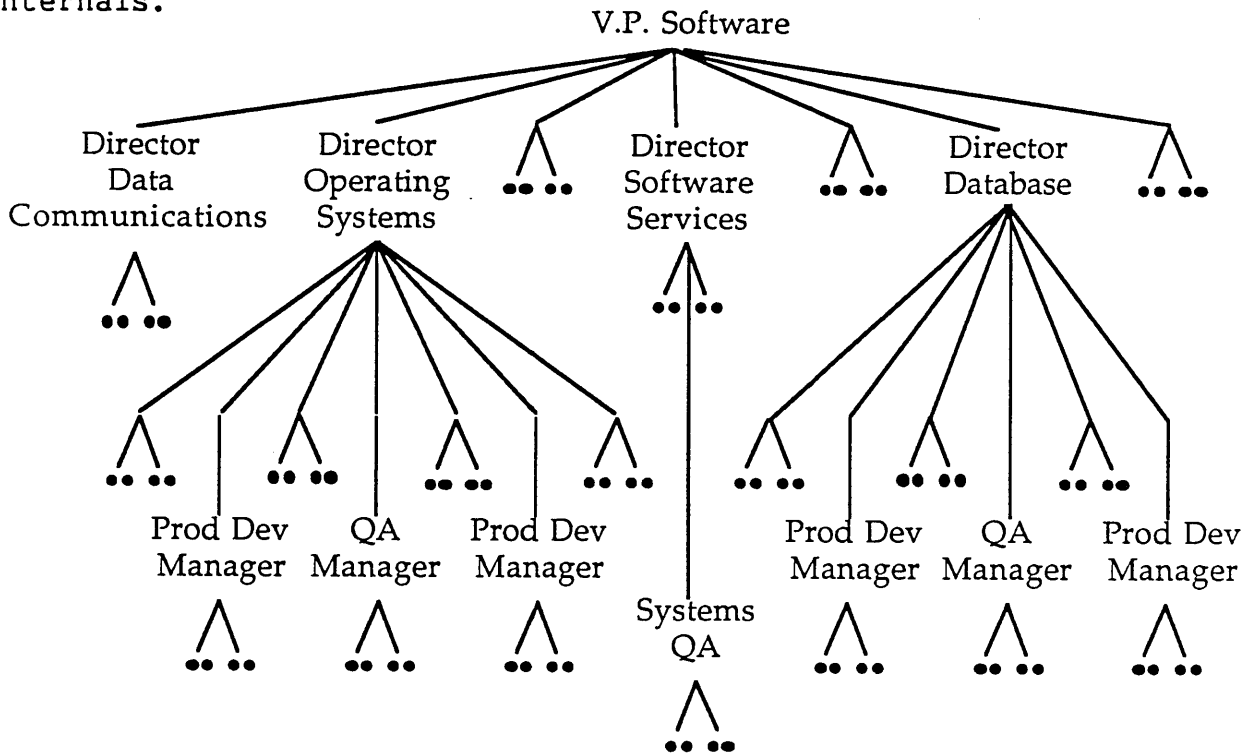
Tandem has faced the classical tradeoff between isolating the testing organization from product development for objectivity, and incorporating the testing organization into product development to find problems early. We have found a balance in today's organization. Software development occurs in three divisions: Operating Systems, Data Communications, and Database software. Each of these divisions has a second line organization devoted to Software Quality Assurance. The distribution of the QA function across the three divisions holds each division Director responsible for the quality of his or her own products. There is also a Systems QA group outside the three product development divisions. Systems QA is responsible for testing the overall software system, as well as testing the migration path from old to new releases.

QA developers have the same job titles and compensation as product developers; we recruit highly skilled, fully competent software professionals into the QA organization. Tandem's structure also fosters a team approach to building quality software; for example the QA developers share office areas with the product developers. This team approach involves QA developers in all stages of the development

cycle, so they can represent the end user by not worrying as much about how hard it is to implement needed functionality, find problems early, and later in the cycle understand product internals so testing will be effective.

The potential disadvantages of close teamwork are the loss of objectivity, and the loss of a QA identity. Keeping QA separate through the second line management level ensures a large enough group (generally 30 to 60 QA developers) to foster a sense of group identity. To ensure cross-department communications, bi-monthly QA managers meetings are held during which overall QA strategies and strategic product issues are discussed. We also hold all-QA meetings, training courses, and social events.

Objectivity is encouraged by writing Test Plans and Test Strategies very early in the development cycle, so that the original thoughts about product testing are captured before learning too much about the internals.



## 2.0 Process

### 2.1 QA Training

In addition to standard internal training available to all developers, Tandem provides QA training for QA developers and interested product developers. We developed a QA course geared towards our development QA environment. In addition, the "Testing Computer Software" course developed by Hetzel and Gelperin has been taught in-house; over half of the current staff of QA developers have attended a course.

Our internal course lasts a week with long lab sessions. It borrows from the IEEE seminar on the Software Test Documentation standard [IEEE84]. Students learn about Tandem's QA tools and procedures. They review a small product requirements document and an external specification, generate a test plan and test design specification, and implement a representative sample of test cases.

We make available to all of our QA people the *IEEE Standard for Software Test Documentation* [IEEE83], and *The Complete Guide To Software Testing* [HETZ84]. In addition the Tandem library has many of the standard works on QA. We also have an on-line bibliography of recommended QA readings for those interested.

## 2.2 Philosophy

Tandem's philosophy is to provide guidelines, rather than to mandate policies, wherever possible. This practice is consistent with the corporate strategy of hiring the most highly qualified people available for every position, then giving them authority and responsibility for their projects. A project is usually two to twenty people. Thus, rather than having a mandated Software Development Life Cycle Policy, we describe our general method of product development in the Software Projects Handbook. Management decides the lifecycle and the variations from the guidelines for each project.

The Handbook helps project leaders run successful projects, where success is defined as (1) yielding a quality product, (2) on time, and (3) with minimal frustrations by the project team. The Handbook describes the software development cycle in general, as well as variations for different types of projects, such as new development, maintenance, etc. The Handbook also contains a section of essays, or "how to" aids, written by experienced software development personnel. A new product development project leader, working with QA and Publications, will define a custom life cycle for the project, based upon the sample development cycles listed in the Handbook. Below we describe a typical product development cycle.

### 2.3 Product Requirements

The first document produced in the product development cycle is the Product Requirements document (PR), which describes what need the product will fill. The PR, typically five to thirty pages is written jointly by development and product management, and describes the market needs, discusses competitive products (if any), and assigns project priorities, such as schedule, performance, and functionality. The PR is submitted for general distribution, with review comments submitted by development, QA, publications, marketing and support. Tandem has no formal standards for writing this document, although there are guidelines and examples of previously written documents.

### 2.4 QA Strategy Document

This document gives a high level overview of QA's strategy for an upcoming product release. Keeping in mind that a release can mean the development of an entirely new product, or an enhancement of an existing product, the QA Strategy document is organized as follows. First there is an introductory section, which describes what is new or changing for this release. Then, the impact of these changes on product quality is discussed, as well as an assessment of the risks. Next, QA's strategy for managing the risk is described. The specific steps QA will be taking to contain the risks and why these actions were chosen are discussed. Constraints, work-arounds and our assessment of the quality of these decisions are also outlined. This document must be concise to ensure that it is widely read; in general, it is one page long. Review is done by product development, QA, product management, publications, and support.

## 2.5 External Specifications

From the PR, product development writes the External Specification (ES), which describes the product as it will look to the user. A single ES is usually thirty to one hundred pages, although some have been over two hundred pages. All products must have an ES. The ES is widely distributed for review. In particular people from product development, QA, publications, marketing, support, and education review the ES. Many of the reviewers use the ES as the basis for their own documents relating to their role in the project. There may be several iterations of reviews if there are enough comments or changes to warrant them.

## 2.6 QA Test Plans

Based on the first ES, QA developers write a Test Plan (TP). Tandem generally follows the IEEE standard *Test Plan* as defined in the *IEEE Standard for Software Test Documentation*. Writing the test plan encourages a thoughtful review of the ES. The test plan, typically ten to thirty pages, is distributed for review when the ES, after one or more reviews, has stabilized. Test plans are reviewed by product development and QA.

The TP may also define what product development must do to show QA they have tested their product. This may entail development demonstrating the passing of a subset of the QA written tests showing basic functionality and demonstrating a specified level of code coverage. QA does not accept a formal release from development until the acceptance criteria have been demonstrated.

## 2.7 Internal Design

The goals of the internal design phase of a project are to produce a design that fulfills the product requirements and external specification, communicates information about the product internals to other team members, and can be used as an aid in maintaining the product in the future.

Early stages of internal design overlap the ES review cycle. The standards for documenting and communicating internal designs are established on an individual project basis; however a formal internal design specification is seldom written. Design walkthroughs, memos, discussions, and extensive comments imbedded in the source are all used. In some cases, a more formal approach has been followed; an example of this approach is an internal design that uses EXCELERATOR [INDE85], a System Analysis and Design program.

## 2.8 Product Development Test Plans

Concurrently with the internal design phase, product developers are encouraged to write a formal product development test plan describing the planned module and integration testing of their product. More often the approach taken to development testing is informal.



## 2.9 Code and Test

Overlapping the internal design phase is the actual coding of the product. Tandem has not imposed any development-wide coding standards, although many project teams have created their own. Reviews are held of selected portions of the internal design and code. These reviews follow a variety of formats, ranging from code reading and reviews to more formal walkthroughs.

After coding, product development performs module and integration testing. Results from this testing, as well as information gathered during reviews, guide QA in the development of test library enhancements that target potential product weaknesses.

### 2.10 Test Design Specifications

QA uses the ES and their TP as the basis for writing their Test Design Specification (TDS). This document is an amalgamation of the IEEE standards for *Test-Design Specification* and *Test-Case Specification*. Most of the test cases described in the TDS have the same test items, hardware, and software. By listing all the test cases in one document, we avoid a lot of repetition. A TDS is typically thirty to a hundred pages long.

During work on these specifications, we often discover areas in which the product's ES is incomplete or ambiguous. The QA developers request a revised ES. QA might also ask product development for special test points to be inserted into product code to guarantee the

software is in certain states when a test performs certain actions. This can be especially important in testing software fault-tolerant operation [BART86] where it is important to verify recovery logic.

If not already defined in the TP, the TDS specifies the release to QA acceptance criteria. The TDS is reviewed by product development and QA.

### 2.10.1 Test Library

The TDS describes the test library to be built. All tests become part of the automated regression test suite. Most libraries are designed to permit the later addition of other types of test cases, for example internals based tests, or test cases based on a specific problem reports.

In the design of the test library, the following points are always emphasized:

- Each test should have a specific objective as defined in the test case design. Tests must check for correct (postive tests) and incorrect product usage (negative tests). There must also be stress, load, fault-tolerance, and performance tests.
- Anyone should be able to tell whether the software passed or failed a test. The test results must be complete and consistent. In the case of failures, enough of the failure environment must be retained to allow diagnosis.
- The tests should be runnable by anybody. To ensure portability,

tests must include parameters to allow them to run in different environments. Each test should be self-contained. It should not depend on other tests.

- Tests should be automated as much as possible, using the tools described later in this paper.

### 2.11 Test Cases

From the TDS, QA codes its test cases. The first test cases written are generally based on product externals, since internals might not even exist yet. Sometimes there are test case walkthroughs, but they are still rare.

Associated with the test-case development is the writing of test-procedure specifications, which describe how to run the tests. In concept these specifications are like the IEEE's *Test-Procedure Specifications*, but the IEEE outline is not used. All together, the test-procedure specifications may be many hundreds of pages.

### 2.12 Developer Release

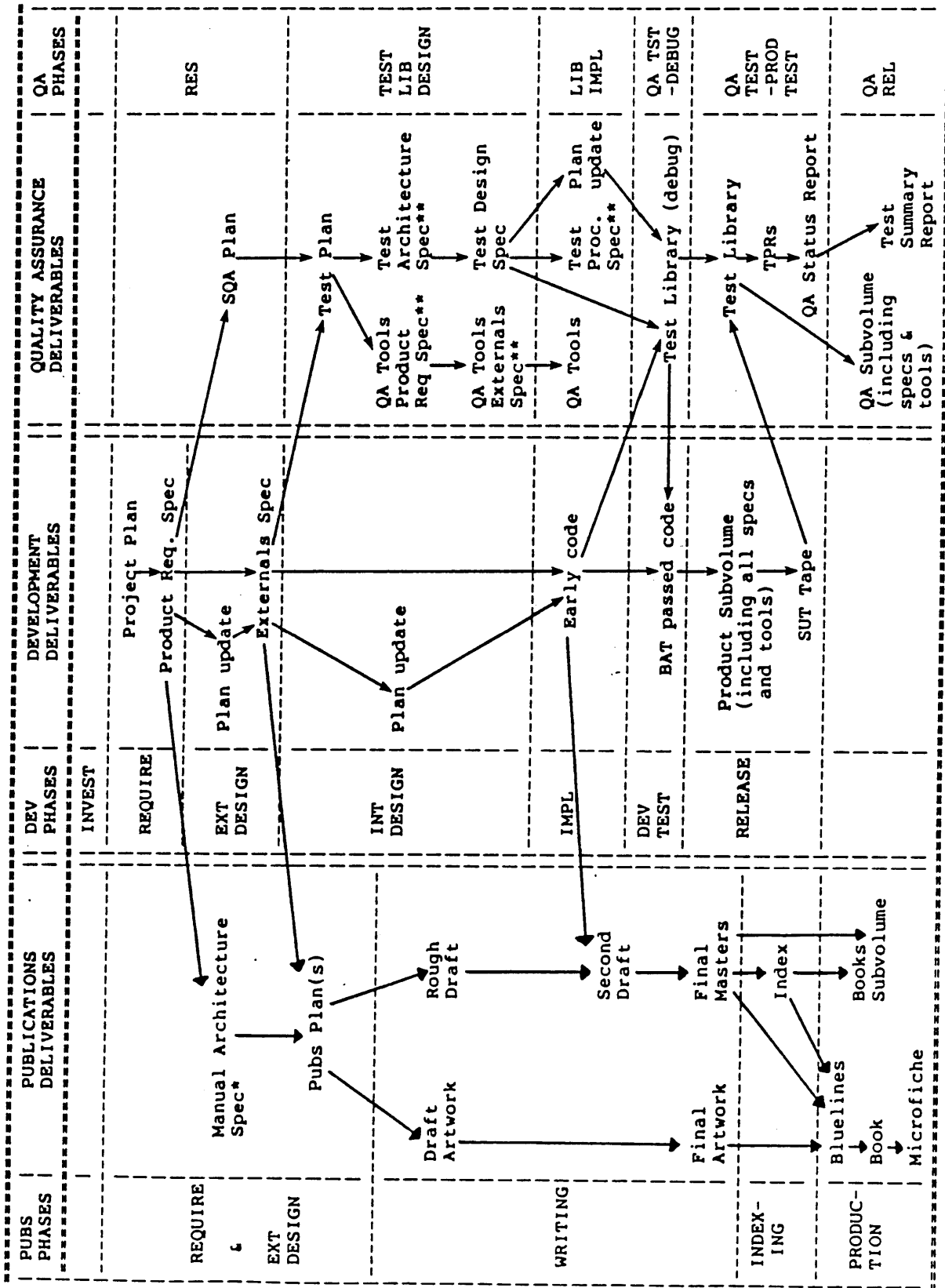
As product development completes its module and integration testing, developers give QA pre-release versions of software for test-case debugging. Problems found while debugging test cases are reported to product development. QA might also begin implementing internals and coverage based testing at this point. QA enhances the TDS to discuss the new tests and new types of problems uncovered by the test debugging.

Product development works toward passing the acceptance criteria defined in QA's TP or TDS. Product development and QA agree to these criteria early in the development cycle. The criteria ensure that a product formally released to QA by product development has a known level of functionality and stability.

### 2.13 Release Build

After meeting the acceptance criteria defined in the TP or TDS, product development formally builds and releases the product to the release coordination group using Tandem's automated release tools, causing creation of Tandem's equivalent of the IEEE *Test Item Transmittal Report*. The release coordination group provides the software to QA test machines for formal testing. QA operations checks out the integration of the software by installing all of the official software together. QA tests the official, controlled software running only with other official, controlled software and tracks problems. QA also measures coverage (see COVER later) of the official product.

The figure on the following page is from the Handbook and shows the various phases and deliverables.



The Big Picture

(\* only if multiple manual project) (\*\* may be included as sections of the Test Plan)

## 2.14 Manuals

The writers of Tandem's manuals are also within the Software Development organisation. A publications plan is distributed for review shortly after the ES is finished. The plan is reviewed by publications, product development, and QA. The writers strive to have draft copies of the manuals ready during the initial testing period of a release. Both product development and QA review the manuals. QA also tries to start using the manuals to verify their ease of use as well as their technical accuracy.

## 2.15 Classes

From the ES and draft copies of the manual, Tandem education prepares classes for Beta customers (described later) and field analysts. A dry run of the class is given to product development, QA, education, Alpha users, and others.

## 2.16 Systems QA

The focus of Product QA is on testing individual products. Within software development there is a performance assurance group to identify performance problems in individual products. Large-scale tests as well as testing of migration to the new release are the responsibility of Systems QA.

Systems QA is organizationally separate from Product Development, reporting to the Vice President of Software through the Director of Publications and Software Services. Systems QA begins working with the software midway through the formal release testing cycle, acting as the first customer-oriented user of the system. Systems QA

consists of a core group of QA developers and analysts who, with the help of visiting field analysts, concentrate on the systems testing of releases. Systems QA uses copies of large customer applications that have proven stressful to the system. Visiting analysts may bring new applications for testing, along with new ideas about specific system tests to try. Large-scale test cases are custom designed for each major new software release. Large hardware configurations are used to discover any unknown systems limits. Almost every hardware device type Tandem currently supports is available to Systems QA to ensure compatibility. Systems QA also verifies the installation procedure, and performs extensive power-fail and network testing.

Outside of software development, the major test groups are within the Customer Support Organization. These groups review classes, manuals, etc. They prototype customer applications as part of application design reviews. They verify connectivity of data communications products with other vendors' products, especially IBM SNA. The Benchmark Center conducts performance analyses of customers' applications. The high-performance center runs the system at peak performance, looking for bottlenecks, performance bugs, etc.

### 2.17 Alpha Testing

The intent of Alpha testing is to provide testing in a real user environment with particular emphasis on functionality, stress, and volume. Alpha testing occurs concurrently with pre-release testing, once most of the serious problems have been found and corrected during the product QA cycle. Alpha testing is done on most of our development systems and some corporate systems, which are all large systems with many users. Problems found during alpha testing are

reported and tracked, as are those found by product QA. The success of alpha testing, and the rate at which problems are found during alpha testing, are factors used in determining the product's readiness to ship to Beta customers.

## 2.18 Beta Testing

Beta customers are the software's first non-Tandem users. Beta sites serve a dual purpose:

Validation - How well does this product meet the customer's needs?

Verification - Is the product correct / defect free?

Beta customers agree to use the product in a non-production environment, and are aware that at this stage the software may still be changing. The Beta testing process is tightly controlled, starting with the selection of Beta customers who are willing and able to give the most useful feedback about the product. Beta customers attend training courses, receive draft copies of manuals, and are monitored closely by field analysts to help ensure easy usage of the product. Beta Customers work with their Tandem field analysts to produce Beta test plans, which are reviewed by product development, product management, support, and QA. During actual Beta testing, progress against these plans is closely monitored, both by the analyst and through weekly status reports submitted to product management. Refreshes of Beta software are scheduled as needed.



## 2.19 Controlled Distribution

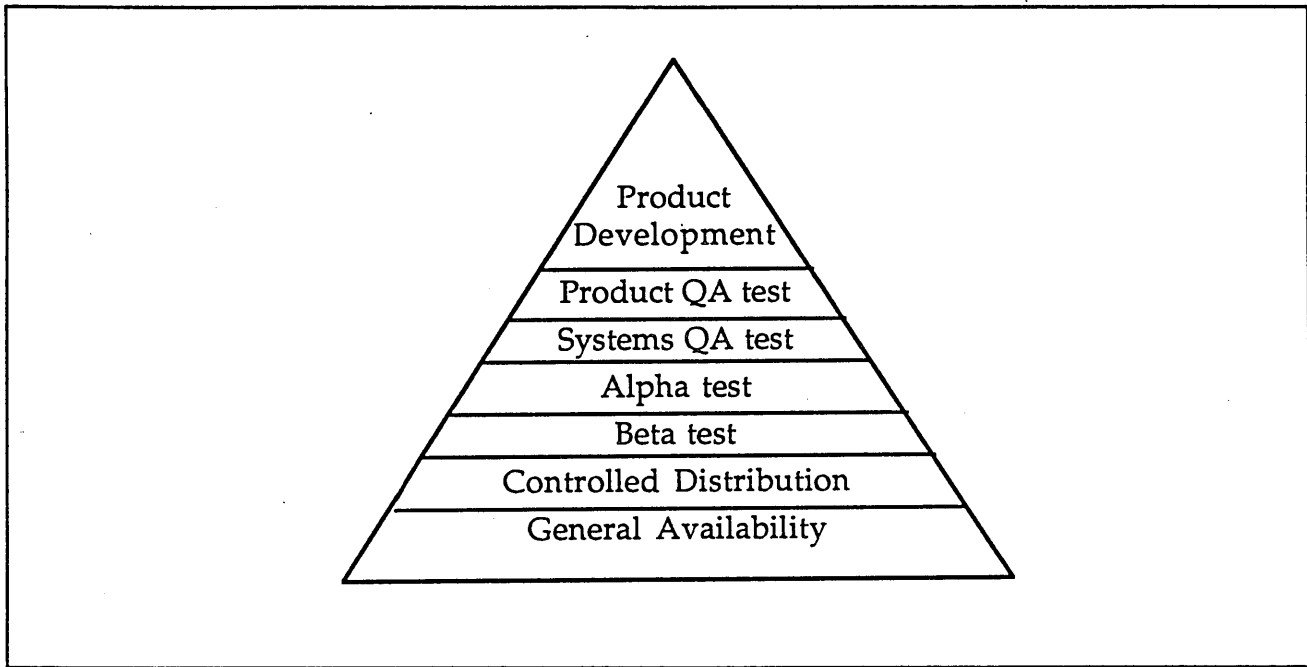
Once Beta testing has proved customer acceptance of the product and product and systems QA have given approval, the software may enter into a controlled distribution situation. This option is most often chosen for a large new product, or for a major systems revision. By closely controlling both the number and type of customers who receive early versions of the product, we can provide close field analyst support as well as support from development. As a Limited Customer Distribution is considered a release of the product, we charge for the products. Customers may use the product in a production environment.

## 2.20 General Availability

At this stage of the product's distribution, the product is considered to be generally available to anyone who is interested in purchasing it. The initial problems encountered by customers, both during Beta testing and during limited distribution, have been addressed. The support organization has been trained and is in place. Training programs have been developed and customer tested, and are now ready to be offered on a regular basis. Update releases are scheduled every four to six months to provide bug fixes and planned enhancements to the product. Both problems found and suggestions for improvements and enhancements are tracked via the Product Reporting System (described later) database. In cases of extreme customer problems, Early Warning Reports (EWRs) are generated, and producing responses to these are given a very high priority in development. If a problem is deemed too severe to wait for the next scheduled update release, an Emergency Bug

Fix (EBF) release is performed. Depending on the nature and extent of the problem, and the risk associated with the fix, an EBF can be distributed to all customers or sent selectively.

The following figure shows the controlled growing use of a product or release.



### 3.0 Tools & Measurements

Tandem has built many tools to automate the production and testing of software. Tandem's systems use a proprietary instruction set, architecture, and systems programming language, so most tools have been developed in house. We believe our tool set is generally keeping up with the state of software QA technology, ahead in some areas and behind in others.

Below is a brief overview of the tools discussed in more detail throughout the rest of this section.

ALIEN	Test library management Test execution tracker
QACOMP	Pattern matching file comparison
DIVER DYNREL	Halt a processor Automatic Dump and Reload of a system
TSIM	General purpose Tandem terminal emulator
JET	PC emulator with interfaces to Tandem systems
VC	Version Control system
PRODRLSE	Tracks product transmittals
COVER	Statement coverage analyzer
PRS	Product Reporting System

An individual test case in an ALIEN managed library can run TSIM against a product and use DIVER to test fault tolerance. The results would be verified using QACOMP and incidents reported using PRS.

With all of the various testing efforts within Tandem, it is important to measure the testing to ensure that the test accurately assess the quality of the products. One of the best indicators of the quality of

a test library is how many problems it found before release compared to the number of problems found after release by customers and others. Another indicator of a test library's quality is what percentage of the product is covered by tests in the library.

Tandem's measurements generally provide only raw data. Using Tandem's report writer and personal computer programs, informative reports and graphs are generated. Some analysis of the data still requires manual work. For instance, the overall report on the quality of a release is usually gathered manually since we lack a unified database of test results [LIND86].

### 3.1 ALIEN

Tandem wrote its Automated LIBrary ENVIRONMENT (ALIEN) to help manage test libraries. ALIEN was written in Tandem Advanced Command Language (TACL), an interpretive language similar in concept to the Unix(tm) shell but more powerful.

Libraries managed by ALIEN have two main components: test units and shared units. Test units are independent, separately runnable entities. Each unit can contain many test cases. A test unit has an associated list of keywords with which to select it and a list of tools it requires in order for it to run.

There are two kinds of shared units: shared tools and global databases. A shared tool is common code or programs used in more than one test unit. A test unit can also contain specialized tools that only it needs. A global database contains data for use by two or more test units.

ALIEN keeps track of the state of development of test units and shared units. *Under-development* means the test units have been initialized. A test developer creates and tests a test unit while the unit is in the *under-development* state. When the test developer is satisfied, he or she checks it in to ALIEN as *Serviceable*. ALIEN archives checked-in test units so that anybody needing to test a product knows where to find stable, runnable tests. When another developer has run the unit, its state can be marked *Certified*, meaning there is a good chance anybody could successfully run the test.

A tester can select all archived tests from the library or only those matching a specified set of keywords. A tester may also exclude tests based on keywords. For instance, the tester can select all STRESS tests that do not require a NETWORK. ALIEN helps install shared units required by the test units that have been selected to run.

Each test has at least three parts:

- setup checks configuration parameters, etc.
- procedure runs the actual test
- cleanup returns the system back to its original state.

ALIEN usually runs on the system where the test is performed. ALIEN also allows multi-step tests. For instance there might be a test in which the first step causes the system to halt; the system is restarted, and the second step of the test runs to complete the test. Multi-step tests verify system failure recovery, for instance recovery from a long power outage.

ALIEN logs the running of tests not *under-development* and can produce reports about test runs. A test always reports one of three results: *Passed*, *Failed*, or *Deferred*. Most tests always report either *Passed* or *Failed*. *Deferred* indicates that the tester must make the final judgement.

ALIEN logs the starting time, stopping time, and evaluation result of each test. Furthermore, the user can set a 'baseline' from which test reports are to begin. For example, the user can request a report on all tests run since the start of a release and a report on all tests run since the latest transmittal of software. Test histories of different transmittals can indicate the relative stability and quality of the product and can sometimes predict the future maintainability of the product. There is currently no provision for combining the results of different test libraries.

### 3.1.1 Example ALIEN reports

STATLIB ->> QA Library Reporting Facility

Test logs used were:

\$SYSTEM.DSTLOG.TESTLOG

-----  
SUMMARY BASELINE REPORT -- 11/23/86 11:01:27 PM

Baseline = FIRST RQAT RUN

TEST UNIT	Number of Attempts	Number Failed	Number Passed
----	-----	-----	-----
T001	1	0	1
T002	1	0	1
T003	1	1	0
T004	2	1	1
T005	2	0	2
T006	2	0	2
T007	2	1	1
	-----	-----	-----
	11	3	8

DETAIL BASELINE REPORT -- 03/17/87 06:21:56 PM

Baseline = NCN-RELEASE OF 1/19/87

TEST UNIT	EVENT NAME	EVENT-DATE	EVENT TIME	EVAL FLAG	EVALUATION-COMMENT
T003	START	3 FEB 87	09:10		
	STOP	3 FEB 87	11:19		
	EVAL	3 FEB 87	11:19	FAILED	CASE 6.5-4: GETVERSION FAILURE:-8
	START	6 FEB 87	10:33		
	STOP	6 FEB 87	10:41		
	EVAL	6 FEB 87	10:41	PASSED	GETVERSION OK
T004	START	29 JAN 87	07:06		
	STOP	29 JAN 87	07:10		
	EVAL	29 JAN 87	07:10	PASSED	SIMPLE FILTERING OK
	START	3 FEB 87	11:20		
	EVAL	6 FEB 87	13:20	FAILED	INSTALLATION PROBLEM
T005	START	29 JAN 87	07:11		
	STOP	29 JAN 87	07:22		
	EVAL	29 JAN 87	07:22	PASSED	SIMPLE REAL TIME CONSUMER
	START	3 FEB 87	11:22		
	STOP	3 FEB 87	11:23		
	EVAL	3 FEB 87	11:23	FAILED	EVENTS NOT FOUND, R1:QACOMP DIFF

The EVALUATION-COMMENT is part of the test output for tests that automatically evaluate to Passed or Failed. For tests that do not complete normally (second run of T004 for example) or have Deferred results, the EVALUATION-COMMENT is entered through a separate ALIEN function to manually evaluate the test. Eventually all tests should be listed as Passed or Failed. In this example, test T003 detected a product failure on the first run but on the second run the product passed.



ALIEN does not track the number of tests planned. The number of planned tests is usually tracked manually based on the TDS.

### 3.1.2 Weekly Release Testing Reports

During the formal release testing cycle, summary reports of testing progress are produced weekly. These reports measure testing progress against the planned schedule. Items tracked are the number of tests planned, available, run, and run without incident (passed). Problems are tracked by total number found, total number of severe problems found, current unresolved problems, and current severe unresolved problems. Graphs of these statistics over time are used as tools to determine how well we are using our existing test libraries, how well we are adhering to the planned schedule, and when the product is ready to ship. These reports also highlight areas of the system that are stabilizing more quickly than others, allowing us to redistribute resources to match the need.

### 3.1.3 Example cross product weekly summary

	Planned	Available	Run	Passed	Pass/Avail
Product set 1	280	238	177	112	.470
Product set 2	55	30	30	29	.967
Product set 3	23	16	16	12	.750
Product set 4	105	105	104	100	.952
Product set 5	440	432	377	361	.836
Product set 6	408	408	363	363	.890
Product set 7	294	150	150	111	.740
Product set 8	461	391	310	295	.754

ALIEN serves a similar purpose to AT&T's BUSTER [WODA86] or Software Research's SMARTS [CASE87]. There are many similarities between ALIEN and BUSTER showing the need of all test organizations for test support systems. Like ALIEN, BUSTER requires test scripts to be independent and its tests have three parts: setup, procedure, and cleanup. A BUSTER test can have any of three results: *pass*, *fail*, and *inconclusive* (equivalent to Deferred). Unlike ALIEN, BUSTER tracks the number of planned tests and it can time out a test.

### 3.2 QACOMP

The Quality Assurance COMPArison tool, QACOMP, is a general purpose file comparison program containing special features to support testing. A tester who uses QACOMP must run the test at least once to create "reference" files. Reference files are carefully checked manually so that subsequent test runs can reliably compare newly generated test results against them.

In large production systems it is rare to have a reference file which matches its actual results file byte-for-byte. The files are often filled with inconsequential differences, such as timestamps, that are irrelevant to the function being tested. QACOMP provides a robust pattern matching facility which permits the tester to specify exactly which parts of a file are allowed to differ.

QACOMP has a report generator that allows the tester to document tests, and have the documentation appear on the report alongside the result of the comparison. The report generator also permits the tester to indicate in the documentation that the reference files aren't quite right yet. This is very useful in situations where low

priority defects still exist and the problems may not be fixed for a while. Thus, if the files match, QACOMP will indicate FAIL/NOCHANGE instead of PASS; a difference in the files will be highlighted as FAIL/CHANGED.

### 3.3 DIVER & DYNREL

DIVER is a program that causes a processor to halt under software control. It runs in the processor to be halted, waits a short random time, and then halts the processor. The halt may occur at any time including catching an I/O process in an I/O for instance. DIVER is used to halt processors to verify fault tolerant software. Loss of a processor should have virtually no effect on system operation. DIVER is shipped to customers so they may conduct their own tests of this essential feature.

DYNamic REload, DYNREL, is a personal computer emulation of a Tandem operations and service processor (OSP). The OSP allows an operator to control Tandem processors, for instance to cold load or run diagnostics. With DYNREL a test can intentionally cause a system failure, reload the system, and continue. A DYNREL script can also cause different system configurations to be loaded for each reload of the system. DYNREL scripts are frequently used in conjunction with an ALIEN multi-step test unit. DYNREL can also be set up to monitor a system so that if a halt occurs, it can dump all the processors, reload the system, and run the next test or rerun the previous one.

### 3.4 TSIM

Tandem's Terminal SIMulator, TSIM, is completely terminal-type independent and can simulate any terminal type supported by Tandem. TSIM records the actual byte stream sent to and received by a terminal. During playback, TSIM replays the byte stream to the application making requests while recording the applications output for comparison. TSIM lets the user have a display terminal show the progress of a playback as it proceeds.

Sessions with multiple terminals can be recorded concurrently including their relative order of keystrokes. In playback, these sessions can be synchronized so that applications see input in the same order as they were originally entered. This feature is useful for testing an application that shares a cache or uses locks.

TSIM has some major drawbacks. It is not easy to update a previously recorded session. You generally need to recreate the entire session to change it. The comparison can be done as the data is received or the data can be logged and compared in batch mode after the test. An error, such as going to the wrong screen, can cause the rest of the test to miscompare.

TSIM is a good basis for a terminal simulator with good performance and low system overhead but requires support code to better control and update previously recorded sessions [CASE86]. TSIM also needs a full screen editor to change data and test flow similar to Test Work Station [PERE86]. (TWS, which runs on personal computers, is strictly 3270 oriented and therefor not general enough for Tandem's needs.)

### 3.5 JET

The JET test harness supports testing of programs running on personal computers. It allows PC programs to receive input from, and send output to, a control process on the Tandem system. Existing Tandem tools can then be used as an aid in the development and execution of tests and in the checking of test results.

JET provides the following features:

- translation of keystroke input into human-readable text files containing multiple test steps;
- supplying keystrokes to the product one step at a time, either as needed or at tester convenience (stepping mode);
- recording screen output after each testing step;
- allowing test developers to modify test step sequences during test execution according to conditional statements inserted by the test developer (loop or branch based on results of previous test steps);
- allowing testing of multiple versions of the program (multiple PCs) simultaneously.

In addition, JET does not alter the testing environment to the extent that the results of tests run under JET could not be relied upon to correctly reflect the product's behavior without JET.

### 3.6 Product Specific Testing Tools

The previously mentioned testing tools are general purpose and may be used in many different test libraries. Similar tools are available from outside vendors. In addition to these tools, there are tools more specific to Tandem's products. Many test libraries also include product specific testing tools. For instance, to test Tandem's product for auditing a database to keep it consistent (Transaction Monitoring Facility - TMF), the TMF QA group wrote a tool that does random inserts, updates, and deletes to a file or set of files. Each transaction changes the database so that the sum of a specific field in all records of a file are always 1,000 times the number of committed transactions. If any transaction completed only part of its changes, the sum would not compare properly. With this tool, SELFCHECK, we can test database consistency under various types of failures. Failures include an aborted transaction, remote system access loss, single processor failure, and total system failure.

### 3.7 VC

Tandem has developed its own prototype software version control system, VC. VC is language independent and can be used with any set of text files. Thus not only the source, but its documentation and other project related files are under version control. VC allows for parallel development on the same file and implements a unique merge algorithm for resolving differences when the parallel developments are integrated. The VC merge algorithm is more likely the other version control systems to produce the desired merged

result by its remembering the history of changes. Many other version control systems perform merge as though the two files have no prior relationship.

VC is currently being integrated with our product build tools to control the entire software construction process.

### 3.8 PRODLSE

After building and testing their product, developers formally release it using the product release program, PRODLSE. PRODLSE handles the transfer of developers' product-related files to the official archive. PRODLSE is integrated with Tandem's electronic mail system providing the equivalent of the IEEE Standard *Test-Item Transmittal Report*. PRODLSE can be used to release just portions of products for bug fixes.

PRODLSE is one of many integrated release tools used. The entire release process is heavily automated to allow careful tracking of released code.

### 3.9 COVER

Tandem has implemented its own unique coverage analyzer called COVER. It measures statement coverage of virtually any object code running on a Tandem NonStop System. Since COVER causes negligible degradation in run-time performance, it can be used to analyze real-time processes such as the disc process or X25 I/O process without introducing timing anomalies. We can use COVER against the same code that is finally shipped.

COVER is integrated into the operating system and works with all current processors in Tandem's NonStop line. Tandem compilers know nothing about COVER and COVER does not instrument the object code file, it instruments only the in-memory object image. To use COVER, a program must be compiled using compilers that provide symbols for Tandem's symbolic debugger. COVER needs the symbol information to distinguish statement starters and in-line constants. This symbol information can be removed from an object file without any change to the object code when the product is shipped.

COVER lets you take measurements from several different processes using the same program file and merge them together. This feature is particularly important for Tandem's implementation of software fault-tolerant process pairs. We measure the coverage in the primary process doing the work, and simultaneously in the backup which is keeping track of the primary. We then kill the primary process and record which code is executed in the backup, which now takes over the workload. Killing the primary in different states may cause the backup to execute different recovery code.

COVER provides several reports from the measurements taken. It can display each statement number and whether it was executed (default), just the statement numbers which were not executed (brief), or the percentage of statements in a procedure that were not executed (summary). It can also mark up the source listing showing which statements were unexecuted (markup).



### 3.9.1 Example COVER summary output

```
Bitmap is from $DATA.COVER.OR1
Program file is $A.QA.PROG dated 02 Jan 1985, 14:56:01

Spaceid 00 word offset 000045 is base of VERSION^A01
* Entry point at word offset 000014 is VERSION^A01
Of      1 statements,      1 (100. %) were not executed.

Spaceid 00 word offset 000164 is base of APPEND^TEXT
  Entry point at word offset 000000 is APPEND^TEXT
Of      1 statements,      0 ( 0. %) were not executed.

Spaceid 00 word offset 000173 is base of OUTPUT^
  Entry point at word offset 000000 is OUTPUT^
Of      6 statements,      2 ( 33.3%) were not executed.

      . . .

Spaceid 00 word offset 005136 is base of GOTO^
  Entry point at word offset 000014 is GOTO^
  Entry point at word offset 000142 is GOTO^SECOND^ENTRY
Of     46 statements,     32 ( 69.6%) were not executed.

Program total:
Of     771 statements,     227 ( 29.4%) were not executed.
```

A QA developer reviews such coverage reports to evaluate areas that may need improvement. COVER is a tool that aids in test development by indicating areas that need evaluation; it does not provide a strict measure of library comprehensiveness since it can not show areas of functionality missing from the product which the library did not detect [BAIL87].

Other coverage analyzers on the market are based on either a hardware monitor or instrumentation of the source code. Both approaches have short comings. A hardware monitor requires expensive hardware for each processor and complex software to interpret the monitored hardware state. Instrumented code provides a great deal of flexibility in terms of measurements and, if done as a preprocessor,

can even be fairly transportable [GIRA86, MILL86]. However, instrumented code may produce side effects related to increased code size, use of memory, or degraded performance. COVER allows us to test production code and perform coverage analysis on that code in a single execution.

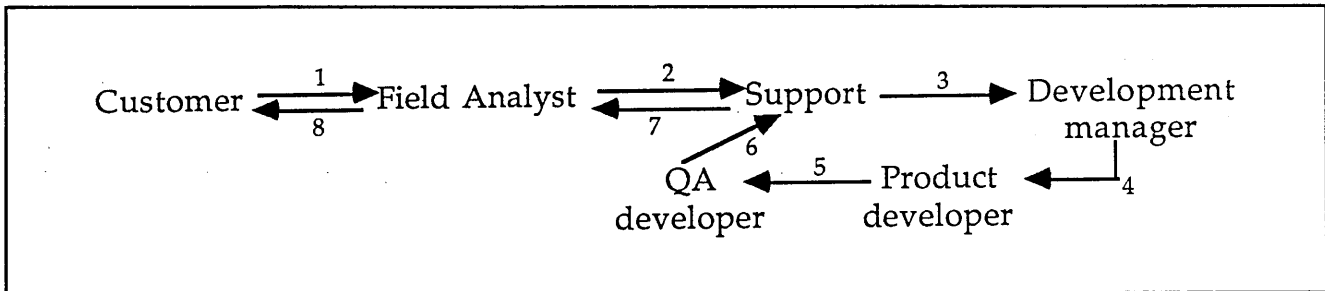
COVER lets us measure statement coverage against real time production code, instead of stronger forms of coverage, e.g. path coverage with instrumented code. COVER can not produce a histogram of usage, but Tandem has performance-measurement tools for determining which parts of code are most heavily used. The performance-measurement tools use a sampling technique that is unacceptable for coverage measurement.

### 3.10 PRS

The Product Reporting System, PRS, is Tandem's on-line tracking system. PRS supports two major entities. A Tandem Product Report (TPR) is equivalent to the IEEE standard *Test-Incident Report*. A Product Maintenance Notice (PMN) is a development designation of the fix. A single TPR can produce many PMNs if it describes many problems, or many TPRs can refer to the same PMN if they are all really the same problem. PMNs can record which module was in error, how much time was spent on the fix, and other development information for tracking. Product developers use PRS to track problems demanding their attention. QA developers track fixes to be verified.

The PRS system is used for all tracking of products that have been released through the release tools. Thus it is used during QA of a new release, from before alpha testing, through Beta testing, and in production. PRS is also the way in which enhancement requests are entered.

A TPR (or PMN) may need to be examined by many different people on different systems. Below is a typical example.



A field analyst enters a TPR on behalf of a customer reporting a problem (1). The analyst assigns the TPR to support for review (2) to see if it is a misunderstanding, already fixed, or something else not requiring development's attention. If support sees the problem as a new one, the TPR is forwarded to development (3). A development manager assigns the TPR to a product developer for response (4). The product developer analyzes the problem and implements a fix. After testing and releasing the fix, he or she forwards the TPR to their QA developer for verification (5). The QA developer gets the fix from the official release archives, verifies it, and returns the TPR to support (6). Support keeps track of known problems to respond to field requests. Support sends the TPR back to the field analyst (7) who informs the customer (8).

PRS causes electronic mail messages to be sent to the responsible people as the TPR is assigned to them. The Tandem report writer can be run against the PRS database to find the current status of TPRs or other TPR and PMN related information.

PRS has a database on each system that uses it. Collecting a complete report involves collecting the information from each system that maintains it. The distributed database can make complete summary reports difficult to acquire and interpret due to the large number of systems with databases.

TPRs are rated on a severity scale of 0 to 3; The most severe problems have a rating of 3 and the least severe have a rating of 0. TPRs can be in several easily discernible locations: Field, Support, Development (includes QA), Publications, Marketing.

The current PRS has several shortcomings as a problem tracking system. It does not keep sufficiently detailed information about the state of a TPR, for example "the TPR is waiting for product developer review" or "the TPR has been sent to QA awaiting release of fix". PRS is also not designed to record categories of problems found as has been suggested by Slingluff [SLIN86]. Many development groups have by convention been able to record this information in various text fields, but it is not yet automatic and nor easy to report.

PRS is currently being redesigned and implemented to provide better tracking of TPR and PMN states within development, as well as to accept other information that is currently manually tracked or tracked via misuse of existing fields.



## 4.0 Conclusion

### 4.1 IEEE Standard for Software Test Documentation comparison

The IEEE Standard [IEEE83] defines eight types of possible documentation for software testing. Not all documentation is necessary for all types of software projects. In some cases we have tailored the standard format to better fit our needs and environment.

Except for the Approvals section, which no Tandem documents contain, many of Tandem's Test Plans correspond exactly to the IEEE's. Our Test Design Specification is an amalgamation of the IEEE's *Test-Design* and *Test-Case Specification*. While most test libraries include test procedure specifications, they follow only the intent of the IEEE and not its form. ALIEN requires a file called DOCUMENT for each test unit; that file corresponds to the test procedure specification. Tandem's equivalent of the *Test-Item Transmittal Report* is the mail message generated by PRODRLSE when a product is submitted. Tandem has very little resembling the IEEE's test log and we are uncertain of its usefulness in an automated test environment where many of the test libraries run overnight, unattended. ALIEN's test log records a test start, stop, and evaluation, but not its "visually observable results" or environmental information. Tandem's TPR/PMN scheme is analogous to the IEEE's *Test-Incident Report* but does not exactly follow the IEEE's outline. Tandem produces weekly summary reports during a release and a final assessment, but not with the formality called for by the IEEE's *Test-Summary Report*. The summary report is an area where we can probably improve.

The table below estimates the percentage of compliance between Tandem's documents and the IEEE software test documents. "Syntax" indicates to what extent we use the titles, subtitles, etc. that the standard recommends. "Concept" indicates to what extent our documents correspond to the spirit of what the IEEE document calls for.

IEEE	Tandem	Syntax	Concept
Test Plan	Test Plan	95	95
Test-Design Spec Test-Case Spec	Test-Design Spec	50	80
Test-Procedure Spec	Test-Procedure Spec	50	80
Test-Item Transmittal	PRODRLSE mail message	40	70
Test Log	Test Log	30	50
Test-Incident Report	TPR/PMN	80	100
Test-Summary Report	Test-Summary Report	0	40

#### 4.2 FUTURE DIRECTIONS

We have recently completed a cycle of improving Quality Assurance at Tandem. This has resulted in a better understanding of our development process, a fuller complement of tools, and raising the level of experience of people in our testing organization. We need to continue work in all these areas, of course, but we also need to obtain better measurements, so that we can understand where best to apply our next efforts at improvement.

Our measurement tools are improving, as well as our ideas as to what should be measured. We are looking for trends among our measurements of what has occurred which will tell us what techniques have worked well in the past, and what are the early indicators that a product is developing into a quality product as well as the indicators that

something is amiss. It is our hope and firm belief that we can continue to improve our ability to deliver high quality software on competitive schedules, and for competitive development costs.

#### 4.3 SUMMARY

Different QA groups are implementing these processes, tools, and measurements at different rates. No single project currently does all that we have described, but we are all striving to achieve the highest levels of software quality.

This paper has described the process followed by Tandem Computers today to ensure delivery of high quality software on schedule. It has discussed our development organization, our internal training, and the Software Development Life Cycle. It has described the tools we use today to enhance testing productivity and to measure control the effectiveness of testing. The goals of our tools are to automate the testing process, track progress and problems found, and measure the results.

#### 4.4 Acknowledgements

The authors wish to thank Bruce Bailey for COVER, Doug Chorey for the Software Projects Handbook, Claude Fenner for QACOMP and ALIEN, David Hammer and Ian Earnest for DYNREL, Ken Franklin for PRODRLSE, Robin Glascock for JET, Jim Mead for TSIM, Chris Sheedy for VC, and Edith Reisner, Tom VanVleck, Howard Moehrke, and our other reviewers for their careful review of this paper.



## REFERENCES

- BAIL87 Bailey, B., *Uses and Abuses of Statement Coverage*, Tandem Technical Report TR87.2, 1987, Cupertino, CA.
- BART86 Bartlett, J., Gary, J., Horst, B. *Fault Tolerance in Tandem Computer Systems*, Tandem Technical Report TR86.2, 1986, Cupertino, CA.
- CASE86 Casey D., Ceguerra L., Irwin J., Morrison M., *User's Manual for CAPBAK, Release 2.0.8*, Software Research Associates TN-1075/8, 1986.
- Capture/Playback system for PC's.
- CASE87 Casey D., *SMARTS - Software Maintenance and Regression Test System Function Description and User's Manual*, Release 3.1, Software Research Associates TN-1281/3.1, 1987.
- GIRA86 Girard, G., *Measuring the Effectiveness of Testing*, Third National Conference on Testing Computer Software Conference Notebook.
- HETZ84 Hetzel, W., *The Complete Guide to Software Testing*, Wellesley, MA: QED Information Sciences, Inc. 1984.
- IEEE83 *IEEE Standard for Software Test Documentation*, ANSI/IEEE Std 829-1983.
- IEEE84 *IEEE Standards Seminar - Software Testing Workshop*, 1984.
- INDE85 Index Technology Corporation, *Excelrator Users Guide*, Index Technology Corporation, Cambridge, MA, 1985.

LIND86 Lindgren, A. L., *Testing Quality Control of Large Software Systems*, Third National Conference on Testing Computer Software Conference Notebook.

MILL86 Miller, E., *Using Tools to Improve Test Effectiveness*, Third National Conference on Testing Computer Software Conference Notebook.

Discusses TCAT an instrumentation coverage analyzer.


PERE86 Perelmuter, I. M., *Direction of Automation in Software Testing*, Third National Conference on Testing Computer Software Conference Notebook.

Describes LEAP & TWS (Test Work Station)

SLIN86 Slingluff, M., *Test Tracking Systems*, Third National Conference on Testing Computer Software Conference Notebook.

WODA86 Wodarz, D., *An Automated Solution to Many Common Testing Problems*, Third National Conference on Testing Computer Software Conference Notebook.

Describes BUSTER.

Distributed by  
 **TANDEM** COMPUTERS  
Corporate Information Center  
19333 Valco Parkway MS3-07  
Cupertino, CA 95014-2599

