A *Roadmap* for Testing

Created/Revised by: S. E. Snook, July 22, 2003

Purpose & Scope:

This document provides a comprehensive, high-level, ***Roadmap*** summary of testing technical guidelines, checklist items, and brief documentation templates. The testing technical data included herein is meant to be a menu of items to select from based on the unique specifics of a given… ***testing trip*** (project). It is not intended to be applicable in totality to any given project. However, it is recommended that all included items should at least be considered/analyzed for applicability based on the specifics of a given project. There should be some sound, technically defensible, logic or reason for not including an item. It should be noted that project **risk**[16&17] may increase in some proportion to the items not selected…*you may get lost somewhere along the way of your testing trip.*

This ***Roadmap*** is meant to be useful for all sorts of ***testing trips***…from a simple, ***eXtreme***[9,26&27]***-ly short walk-around-the-block-testing-trip*** to a ***Biblical-40-year-testing-trek-through-the-desert*** constrained by ***stone tablet regulations***….

Examples:

1. For ***eXtreme***[9,26&27] testing ***walk-around-the-block-trips,*** *extreme* developments, and small simple projects / systems…the recommended, cost effective, and risk-appropriate[16&17] testing activities may be narrow in scope and described by a single paragraph or page. As an *eXtreme* minimum example, Dr. James Whittaker[4] recommends little or no formal test documentation at all, but instead recommends developing a *Fault model*[4] of your *application under test*. Then … *Brain on…eyes open…TEST!*[4]…using various *Attacks*[4] (listed below).

2. For ***Biblical-40-year-testing-treks-through-the-desert*** constrained by ***stone-tablet-regulations***…large complex projects…essentially the opposite end of the spectrum of ***eXtreme testing trips***…one can take a ***Biblical and Regulatory*** approach to test activities and documentation:

   Biblical - Test everything. [1 Thessalonians 5:21 NIV] - A worthy *spiritual* goal, but in general not possible in this lifetime…but one can focus on progress, not perfection.

   Regulatory - If it isn't documented, it didn't happen.[*Food &Drug Administration*] All test activities are completely documented. Each of the technical testing activities may be separate or multiple documents/volumes (e.g., a Test master plan with subordinate multiple testing plans, designs, cases, procedures, and reports).

Experience Note: As effort intensive as it may seem, this ***Biblical and Regulatory*** approach may be somewhat appropriate for high risk systems such as weapons/combat systems, biomedical systems, nuclear reactor control systems, and spacecraft control systems.

Generally however, most ***testing trips*** are of some middle length/size, complexity, and risk[16&17] requiring a somewhat less than ***Biblical and Regulatory*** approach. That is, the scope of activities and extent of formal documentation should be proportional to the project / product size, complexity, and risks[16&17].

An attempt has been made to include a broad scope of summarized, current, testing technology. More detail is provided in the included references. That is, this document is applicable to projects / products of all sizes including initial/new development testing as well as the testing of subsequent computer system software revisions / releases. For a completely new release, this document may be analyzed for applicability to all life cycle phases of development; or, in part for minor releases.

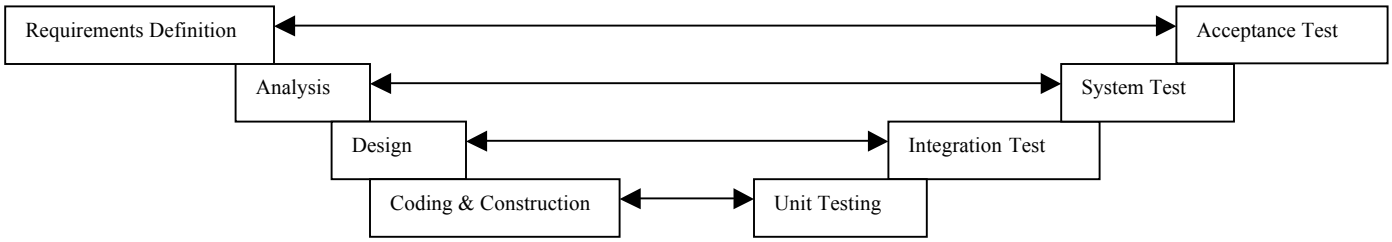
Experience Notes:

1. You never know what is enough [testing] unless you know what is more than enough.
2. To err is human. To really foul things requires a computer.
3. Great engineering [testing] is simple engineering. - James Martin
4. Myth #1: The computer only does what you tell it.
5. There's always one more bug.
6. The Definition of an Upgrade: Take old bugs out, put new ones in.
7. Computers make very fast, very accurate mistakes.
8. "It did what? Well, it's not supposed to do that."
9. "My software never has bugs. It just develops random features."
10. If debugging is the process of removing bugs, then programming must be the process of putting them in.
11. The sooner a bug is found and fixed, the cheaper.[12]

A *Roadmap* for Testing

Created/Revised by: S. E. Snook, July 22, 2003

The "V" Testing Concept[9&15]



In a lifecycle approach to testing, testing should occur throughout the project lifecycle; from testing requirements through conducting user acceptance testing. E.g., looking at the figure above, User Acceptance Test activities should begin concurrently with Requirements Definition activities. The arrows in general show the time relationship of testing and development activities. The earlier an error is detected the less expensive it is to correct. -Barry Boehm, 1976

Testing (Validation) Principles:

1. The primary role of software testing is two-fold: determine if the system meets specifications (developer view); determine whether the system meets operational business and user needs (sponsor/customer/user view)[9] In addition, there are several secondary roles for testing in an organization that include: raising issues; instilling confidence in the system; providing insight into the software delivery process; continuously improving the test process.[9]
2. Testing is a detection activity. It can only demonstrate the presence of errors. Testing cannot prove the absence of errors. Thus, testing is the process of analyzing a software item to detect the differences between required outputs (or conditions) and actual outputs (or conditions). You can't test quality into software - Testing will not improve bad coding techniques or poor coding structure.[9]
3. Present test results as observations…not judgements. This approach goes a long way in developing a congenial constructive communication with the developers.
4. As a general testing principal, all requirements should be validated by one of the following methods: testing, inspection, demonstration, and analysis. This document focuses primarily on the testing method of validation. Some requirements may not be able to be cost-effectively validated by testing. In such cases, requirements should be validated by some other method than testing.
5. Project interim deliverables/documentation may be verified to provide in-progress quality assurance of successful system/software validation. See examples below.
6. Test planning begins as soon as requirements are established, and must be modified whenever the requirements are modified.
7. The testing effort is planned under the assumption that errors will be found.
8. The testing process is documented (plans, design, cases, procedures, and reports) and approved.
9. Every test case/procedure/result should be repeatable.
10. All hardware or software changes should be tested prior to releasing the system to the production environment. This includes regression testing of all software components that could be impacted by the change(s).
11. Testing standards should be established to address: Documentation, Deliverables, Tools & techniques, Quality requirements
12. Software developers are not the only ones to test the software. Independent testers/users (independent of the engineer responsible for development) may be used conduct system, acceptance, and other types of testing.
13. Testers never assume that the system/software is correct. The software testing effort provides that evidence.
14. Start early. Even though you might not have all of the details at hand, you can complete a great deal of the planning effort by starting on the general and working toward the specific. By starting early, you can also identify resource needs and plan for them before other areas of the project subsume them.[9]
15. Keep the test plan flexible. Make it easy to add test cases, test data, and so on. The test plan itself should be changeable, but subject to change control.[9]

Testing (Validation) Principles. (continued)

16. Frequently review the test plan. Other people's observations and input greatly facilitate achieving a comprehensive test plan. The test plan should be subject to quality control just like any other project deliverable.[9]
17. Keep the test plan concise and readable. The test plan does not need to be large and complicated. In fact, the more concise and readable it is, the more useful it will be. Remember that the test plan is intended to be a communication document. The details should be kept in a separate reference document.[9]
18. Calculate the planning effort. You can count on roughly one-third of the testing effort being spent on each of the following test activities: planning, execution, and evaluation.[9]
19. Spend the time to do a complete test plan. The better the test plan, the easier it will be to execute the tests.[9]
20. Always write down what you do and what happens when you do exploratory testing.[12]The primary objectives[9] of testing during the requirements phase are to:
    - Determine that the requirements fairly represent what the is needed
    - Determine that the needs have been defined and documented
    - Verify that a reasonable and valid cost/benefit study has been performed
    - Determine that the business problem has been solved
    - Verify that the control requirements have been specified
    - Verify that a reasonable process was followed in developing the business solution
    - Verify that a reasonable alternative was selected among the most probable alternative solutions"
21. *eXtreme Testing* [9] - The process of testing software in a dynamic environment whereby the program is executing while testing is underway to identify defects and enhance improved quality and productivity. The methodology combines the best practices of traditional software testing methods and processes with the ability to apply these time-tested methods in a rapid development paradigm. It focuses on collaboration between developer, customer, and tester, testing is accomplished in shorter timeframes with an emphasis on developing the software correctly the first time, thereby eliminating the need for lengthy test cycles and redundant tests. This is a highly customizable approach to testing software and is adaptable to any software development environment. The best success is achieved in organizations that already have mature development and testing processes and have attained a level of capability beyond simplistic means.

Levels / Categories of Testing:

For a specific project, an analysis should be accomplished to determine all appropriate/applicable/cost-effective levels of tests. Those selected should be documented along with rational/justification. The following is a partial list of important levels of validation/testing that, as a minimum, should be considered for applicability to a specific project or product:

1. Functional testing (Generally required for all products. The purpose of functional testing is to reveal defects related to the product's/component's functionality and conformance to documented functional requirement specifications.)
2. Unit testing - usually accomplished by developers; Computer Software Unit[1&10] testing
3. Structural testing[9]
4. Exploratory testing (Always write down what you do and what happens when you run exploratory tests.[12])
5. Component/Sub-component testing Computer Software Component[1&10] testing.
6. Walkthroughs, inspections, desk-checking[9]
7. Verification (e.g., reviews, examinations, walkthroughs, desk-checking, or inspection of interim work products such as requirements, design, specifications, documentation, prototypes, code; early detection of errors are highly cost effective.) [1, 2, 9, &13]
8. Developmental integration testing [9&13]
9. Developmental system testing [13]
10. User acceptance testing (Generally required for all products. The purpose of acceptance testing is convincing the user that the product fulfills expected user needs.)[13]
11. Performance/load/stress testing [9&12]
12. Security/access testing [9&19]
13. Usability testing[16, 22,23&24]
14. Operational procedure documentation verification[11]
15. Regression testing (Reliability)[9&12]

A *Roadmap* for Testing

Created/Revised by: S. E. Snook, July 22, 2003

Levels / Categories of Testing: (continued)

16. Alpha & Beta Testing[12&13]
17. Smoke Test - …establish that the system is stable and all major functionality is present and works under 'normal' conditions.[13]
18. Pilot testing[10]
19. Recovery testing - …can involve the manual functions of an application, loss of input capability, loss of communication lines, hardware or operating system failure, loss of database integrity, operator error, or application system failure.[9]
20. Operations testing / Usability Testing (Ease of Operations)[9]
21. Compliance testing - …verifies that the application was developed in accordance with information technology standards, procedures, and guidelines.[9]
22. Manual-Support Testing - Systems commence when transactions originate and conclude with the use of the results of processing. The manual part of the system requires the same attention to testing as does the automated segment.[9]
23. Intersystem [interface] Testing[9]
24. Parallel Testing (e.g., matching test results between current live system and new system.)
25. Compliance Testing (Authorization) - Testing should verify that the authorization rules have been properly implemented and evaluate compliance with them. Test conditions should include unauthorized transactions or processes to ensure that they are rejected, as well as ensuring authorized transactions are accepted.[9]

Web Site Specific Testing [9&14]

Testing specific to Web sites should address:

Visual appeal, usability, readability, accessibility	Site navigation and hyperlinks
Accurate, reliable and consistent information.	Correct data capture
Transactions completion	Gateway software ("Common Gateway Interface")
HTML validation / W3C compliance	Alternative Graphics Text
Shopping cart, price, tax, and shipping calculations	Server Load testing
Database / file / table validation, integrity, authenticity of facts provided	Query Response time
Multiple browser compatibility	User/page response time
Recovery after failure/error; fail-over; reliability; availability	Network security
Payment transaction security	Printer-friendly pages
Style guides/sheets	Client versus server validation & functional partitioning
Search engine visibility / meta-tags	Site map
Pop-ups	Audio/video streaming content
Database middleman software	Cookies
Screen size and pixel resolution	Scalability/capacity growth
Lists	Searchable documents
Forms	Frames
Image weight (load time)	Information architecture - chunking
Usability[22,23,24&25]	Data & access security
Site user impact	Backup / recovery

Test Data and Data Verification:

It is important to consider the verification, validity, accuracy, and completeness of any test data. Queries or utilities may be constructed to independently verify the contents of a test data set. Data sets and data verification queries, utilities, etc., should be controlled as configuration items and form part of the system baseline.

Data sets intended for testing use in general may include:

1. "snapshots" of production data
2. data constructed of specific data entries designed to match particular test procedures; specific expected results
3. messages
4. transactions
5. records
6. files
7. scripts
8. automated tool generated data
9. ad hoc key strokes
10. DBMS views
11. Stored procedures
12. Queries

Test (Validation[21]) Design, Methods, & Techniques:

Evaluating a system…to determine if it satisfies specified requirements.[21] For a specific project, an analysis should be accomplished to determine all appropriate/applicable test design, methods, and techniques. Those selected should be documented along with rational/justification. The following is a partial list of various test design, methods, and techniques that, as a minimum should be considered for applicability to a specific project or product:

requirements based functional testing[5,9,11,12,&13]	domain analysis testing[5]
data flow testing[5]	control flow testing[5]
orthogonal array testing[5]	waterfall testing[5]
boundary value testing[5&12]	risk based testing[16&17]
error guessing[12]	test automation[15]
string testing[10]	eXtreme testing[9,26&27]
test attacks[4] (see summaries below)	greybox testing[4&5]
white box testing[5&21] / structural testing [5&21]/ glass box testing[12]	black box testing[5&21] / behavioral testing[5&21]
transaction testing[5]	state transition testing[5]
all pairs / pairwise testing[5]	decision table testing[5]
equivalence class testing[5&12]	translucent-box testing[12]
operation testing[9]	recovery testing[9]
compliance (to process) testing[9]	execution testing (desired level of proficiency)[9]
path testing[12]	error handling / messages testing[9]
business rules decision tables[5]	Inter-systems/interface testing (data passing)[1&9]
manual support (people-computer interaction) testing[9]	fault model[4]

Verification[21] Types & Techniques

Verification methods are used to ensure that interim life cycle items (e.g., software, hardware, documentation):

1. comply with standards, procedures, guidelines, and processes
2. satisfy the conditions imposed at the start of that [life cycle] phase
3. are consistent with items from a previous life cycle phase.

Experience Note: Review early and review often.

A *Roadmap* for Testing

Created/Revised by: S. E. Snook, July 22, 2003

Management Reviews[20]	Technical reviews[20]
Inspections[20&28]	Walkthroughs[20]
Audits[20]	Concept of Operations Review/Walk-through
Requirements Review/Walk-through	Design Review/Walk-through
Code Inspection/ Walk-through	Test Readiness Review/ Walk-through
Requirements tracing[9] (e.g., throughout the life cycle; requirements versus design; requirements versus tests.)	Phase-end reviews[9]
Physical Audit (correct versions of hardware & software components) (**Experience Note:** Before you release the program, you MUST check that the release disk [media] contain the most recent version of every file.[12])	In-Progress Review[9]
Post Implementation Reviews[9]	Peer reviews[9]

Test Attack[4] Summary - User Interface Attacks[4]

1. Apply inputs that force all error messages.	10. Force the screen to refresh.
2. Apply inputs that force software to establish default values.	11. Apply inputs using a variety of initial conditions.
3. Explore allowable character sets and data types.	12. Force a data structure to store too many or too few values.
4. Overflow input buffers.	13. Investigate alternate ways to modify internal data constraints.
5. Find inputs that mat interact and test combinations of their values.	14. Experiment with invalid operand and operator combinations.
6. Repeat the same inputs numerous times.	15. Force a function to call itself recursively.
7. Force different outputs to be generated for each input.	16. Force computation results to be too large or too small.
8. Force invalid outputs to be generated.	17. Find features that share data or interact poorly.
9. Force properties of an output to change.	

Test Attack[4] Summary - System Interface Attacks[4]

1. Fill the system to capacity.	4. Assign an invalid file name.
2. Force the media to be busy or unavailable.	5. Vary access permissions.
3. Damage the media.	6. Vary corrupt file contents.

Testing (Validation) Documentation Content – Test Plan, Test Design & Cases, Test Report/Results:

The recommended basic components described in this section should be included in all documents, unless there is sufficient and acceptable justification any of them in whole or in part to be not applicable. These basic components may have different names, groupings, or sequences in different projects and it may be necessary to determine if there is an equivalent or suitable substitute. Additional information may be added as required.

A practical note from some experts: A test plan [documentation] is a valuable tool to the extent that it helps you manage your testing project and find bugs. Beyond that it is a diversion of resources.[12]

Experience Note: Lots of documents do not make good documentation.

Test Plan Content:

1. **Experience Notes:**
   To fail to plan is to plan to fail.
   A good plan today is better than a perfect plan tomorrow.-Wag the Dog, movie dialog
   No plan survives the departure point; keep the plan current.
2. Introduction: testing policies; testing standards & conventions; purpose of project tests; scope; project testing objectives; system description; references; definitions/terms; assumptions; constraints; testing documentation approach.
3. Project Testing Communication Approach: Communication mechanisms e.g., formal and informal meetings; working sessions; processes (such as defect tracking); tools (issue and defect tracking, electronic bulletin boards, notes databases, Intranet sites); bulletin boards; email; status reports; in-progress review meetings.
4. Testability: Is necessary and sufficient documentation available to plan testing; is the documentation available/provided written with sufficient clarity, accuracy, and detail to allow the system/product/software to be unambiguously tested.
5. Identification of Configuration Item(s) Under Test: e.g., Hardware (e.g., Specialty/Test Equipment, Communication Server, Terminal, Applications Server, Computer, etc. - nomenclature, version, date); Software (e.g., modules, units, systems, sub-systems, CSCI, CSCs, CSUs, data structures - name, version/revision, and date); User Operational Procedures (document name, version/revision, and date)

6. Risks & Contingency Planning*: Identify both project-oriented testing activity risks as well an application-oriented risks. Both types of risks may potentially affect the success of the testing. (*if not already included in project management planning documentation.)
7. Functions/Features/Design to be Tested
8. Functions/Features/Design Not to be Tested
9. Traceability: all requirements/features should be traceable to at least one test procedure; conversely, are all test procedures should be traceable to at least one requirement/feature; this traceability is best documented in a traceability matrix.
10. Test Technical Approach: project test strategy (e.g., risk reduction); testing levels; test design methods/techniques; test tools; test metrics; test automation strategy.
11. Test Data: creation, verification, and use.
12. Test Results Recording, Data Collection, Log, & Reports: e.g., instructions for performing tests, documenting results, reporting and resolving deviations, documentation/log of errors discovered, error corrections, re-testing, sign-off/approval provisions.
13. Test Execution: e.g., important factors that affect test execution; test cycles; test resources
14. General Pass/Fail Criteria
15. Acceptance Criteria
16. Suspension / Resumption Requirements/Criteria: e.g., corrective action taken when problems or errors are encountered may include correct documentation, re-run test, re-code, re-design, regression testing

Test Plan Content: (continued)

17. Pre-Test Needs/Test Environment: e.g., hardware/test equipment & documentation; test support software & documentation; test data files; test tools & documentation (Test driver software may need to be developed or procured to test some applications; test design documentation should address this situation.); description of the test data design/validation; installation instructions; multiple environment migrations (e.g., *development-> test-> live*).
18. Organizational Roles and responsibilities: e.g., groups and individuals executing test tasks, groups and individuals providing test support functions. (*If not already included in project management planning documentation.)
19. Miscellaneous: Test Deliverables*(e.g., plans, design, cases, procedures, reports, logs); Tasks*; Schedule*; Milestones*; Staffing Needs*; Training Needs*; Pre-requisites Tasks* (e.g., completion of interfacing systems) (*if not already included in project management planning documentation.)

Test Design & Cases Content:

1. Introduction: e.g., same as test plan or note exceptions or differences.
2. Test Approach Refinements: e.g., Coverage (operational scenarios, use cases, requirements, design, data domain, source code, executable code, logic paths)
3. Test Automation: e.g., automated test regime/environment, techniques, selected test tools, applicability and of test automation for specific project testing activities, division between manual and automated tests.
4. Test Case Selection, Rationale, Justification, Traceability: e.g., with respect to features or specifications
5. Test Case Identification (for each test case)
6. Specific Pass/Fail Criteria for Each Feature/Function/Scenario

Test Procedures Content:

1. Introduction: e.g., same as test plan or note exceptions or differences.
2. Test Preparations: e.g., test case, procedure, scripts, and test data names/identification; pre-test/set-up procedures; hardware preparation; software preparation; other Pre-Test Preparations; system/software initialization; test inputs/outputs; criteria for evaluating results; assumptions, constraints, dependencies
3. Test Procedure(s) Description: e.g., detailed steps, expected results, actual results/outcome/log; deviations from test procedures; variances, and problems encountered ; test assessment
4. Errors/Enhancements: e.g., bug / incident tracking / documentation / prioritization (errors are inevitable); enhancement / change tracking / documentation/prioritization (changes are inevitable).

Test Report:

1. **Experience note**: Record *observations* of system behavior. Avoid making *judgements* about products or development staff producing those products.
2. Introduction: e.g., same as test plan or note exceptions or differences.
3. Expected results versus actual results: Define any problems; identify the work and input sequences leading to a suspected problem; identify the test case/procedure producing the suspected problem; identify the system elements involved; identify the flow of work and/or data input leading to the condition; identify the tester(s); assure the problem is repeatable.
4. Detailed, specific verification artifacts/evidence/results should be captured that can be compared against equally detailed expected results, rather than a simple pass/fail format. As an example, these artifacts may be generated reports, screen prints, and test database contents. All artifacts should be unambiguously labeled with the test case/procedure number, step number, date, and tester identification. Actual Results Deviations from Test Procedures, Variances, and Problems Encountered
5. Errors/Enhancements: e.g., "bug"/incident reporting; enhancement/change reporting.
6. Summary of Test Activities and Actual Results.
7. Comprehensive Test Evaluation, Notes, and Recommendations

Signatures and Approvals.

As required and appropriate for the specific organization and/or project.

References:

1. *IEEE Standard for Software Test Documentation (ANSI), IEEE Std 829-1998*
2. *IEEE Standard for Software Verification and Validation Plans (ANSI), IEEE Std 1012-1986*. New York, NY: IEEE, September 17, 1992 (update reference when changed)
3. *A Test Plan Guide and Readiness Checklist To Facilitate the Execution of Capacity Tests,* Jose Fajardo, StickyMinds.com Template, Jun 10, 2003
4. *How to Break Software*, James A. Whittaker, Addison Wesley, 2003
5. *Mastering Test Design*, Lee Copeland, (draft manuscript May, 2003)
6. *Test Plan Review Checklist,* Ross Collard, Copyright Collard & Company 2002
7. *SQE Test Plan Template,* Softwqare Quality Engineering, Version 7.0, Copyright 2001
8. *Test Planning Case Study –2,* Ross Collard, Copyright Collard & Company 2002
9. *Boot Camp for Software Testers,* On-Line Training Course, St Petersburg College, Quality Assurance Institute (QAI), Copyright 2002
10. *Managing the Testing Process,* Rex Black, Second Edition , Wiley Publishing, Inc. 2002
11. *Software Testing in the Real World: Improving the Process,* Edward Kit, Addison-Wesley, 1995
12. *Testing Computer Software,* Kaner, Falk, Nruyen, Second Edition, Wiley, 1999
13. *Systematic Software Testing, Craig & Jaskiel,* Artech House, 2002
14. *The Web Testing Handbook, Splaine & Jaskiel,* STQE Publishing, 2001
15. *Software Test Automation: Effective Use of test Execution tools,* Fewster & Graham Addison-Wesley, 1999
16. *How to Conduct Heuristic Risk Analysis,* James Bach, STQE Magazine, Nov/Dec 1999 (Vol. 1, Issue 6)
17. *Approach to Implementing Risk Based Testing,* Sreeram Kishore Chavali , StickyMinds.com Article, Mar 29, 2001
18. *A Test Plan Guide and Readiness Checklist To Facilitate the Execution of Capacity Tests,* Jose Fajardo, StickyMinds.com Template, Jun 10, 2003
19. *How to Break Software, Security*, James A. Whittaker & Herbert H. Thompson, Addison Wesley, 2004
20. *IEEE Standard for Software Reviews, IEEE Std 1028-1997*
21. *IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990*
22. *Homepage Usability, 50 Websites Deconstructed,* Jakob Nielsen/Marie Tahir, New Riders Publishing, 2001
23. *Introduction to Usability Testing, C*heryl L. Nesta, Vanteon, SQE STARWEST 2001 Conference
24. *User Interface Guidelines*, UI Guide, Bruce Durbin, StickyMinds.com User Submitted Article, Dec 24, 2001
25. *Usability Testing, How an observer can correctly interpret and categorize user behavior*, Bob Stahl, STQE Magazine, Sep/Oct 2001 (Vol. 3, Issue 5)
26. *Extreme Testing,* Ronald E. Jeffries, *STQE Magazine,* Mar/Apr 1999 (Vol. 1 Issue 2)
27. *Testing in the Extreme Programming World,* Robert Martin, Object Mentor, Inc., SQE STARWEST 2001 Conference
28. *Leveraging Inspections, What to do with the data you collect,* Ed Weller, STQE Magazine, July/August 2003