



Expert Testing...Real World Solutions

DAST **Diagnostic Approach to** **Software Testing**

A work-in-progress by
Hung Q. Nguyen
Michael Hackett



LogiGear Corporation

551 Pilgrim Drive, Suite A-1

Foster City, CA 94404

Tel.650.572.1400

Fax.650.572.2822

hungn@logigear.com

michaelh@logigear.com

<http://www.logigear.com>

- **Introduce DAST, the Diagnostic Approach to Software Testing.**
- **Share our experiences using this method.**
- **Provide you with a process guideline.**

- **DAST is an evolutionary test planning method that utilizes diagnostic analysis.**
- **DAST involves asking *targeted* questions:**
- Specific questions designed to collect pertinent information for test planning and testing.
- Answers to questions may be collected through:
 - Questionnaires
 - Interviews
 - Design walkthroughs
 - Exploration

- **DAST focuses on the design, compilation, and distribution of *useful, reusable questions* that facilitate:**
 - The definition and quantification of the testing effort.
 - The definition and quantification of testing deliverables.
 - The definition and quantification of information needed to successfully produce quality deliverables.
 - The data collection and bug finding process.

- **DAST enables you to effectively collect information, especially in an environment where product documentation is non-existent or lacks information required for test preparation.**
- **DAST enables you to focus on what needs to be done — and to know what has been completed.**
- **DAST enables you to architect a framework for reusing questions and avoiding redundant work.**
- **DAST enables you to work from an expert position independent of process maturity, people's capabilities, and project stability.**

What does DAST entail?

- **Define and quantify what you do.**
 - Start at the end, not the beginning.
- **Define and quantify what it takes to do your job.**
 - What do you bring to the table? Run a skills inventory check.
 - What do you need from others? Collect this through diagnostic analysis.
- **Identify the controllable variables.**
 - What you can control, you can improve.
- **Identify the uncontrollable variables.**
 - What you can't control, you must manage.
- **Design and compile questions that fulfill your specific needs.**
 - Be specific.
- **Group your questions into categories.**
 - Consider reusability.

- ❖ **It can be used in any environment.**
- ❖ **It offers high “return-on-testing-investment” when:**
 - Product documentation is lacking or non-existent.
 - The testing, QA, and engineering processes are not well established.
 - The scope of testing is not well defined.
 - You must quickly determine test coverage based on known constraints.
 - You need to give advice on testing strategy before committing to actual testing.
 - You need to quickly learn about the system under test.
 - Those asking you to test aren't fully aware of the testing issues.
 - You need to test IMMEDIATELY!
 - You need to make sure that what's been promised can actually be delivered.

❖ Test Planning

- ***Collect information***
- ***Analyze data***
- ***Determine test coverage***
- ***Determine test strategy***
- ***Write a test plan***
- ***Develop initial test cases***
- ***Control test environment***
- Plan for defect management
- Plan for deployment testing
- Plan for post deployment

❖ **Test Execution:**

- *Run tests.*
- *Analyze results.*
- Build regression and acceptance test suites.
- Generate more tests through exploratory testing.

❖ **Product Release:**

- Run quality assessment sessions.
- Make release decisions based on known quality risks.
- Validate quality assumptions.
- Write release reports.
- Assist in building a support knowledge-base.

- **Software testing is part art, part science.**
- Testing standards and methodologies are not widely agreed upon within the community.
- Although standards and methods such as requirements,
- CMM and ISO-9000 are available, they have not been accepted by some aggressive, rapid-development companies.
- **Current “state-of-the-practice”**
- At best, we are sharing our experiences regarding best practices.

- **The problems:**
- Contents of documents are not well defined.
- Coverage of documents are not well defined.
- Cost of keeping documentation up to date is high, and often overlooked, so we stop updating when we decide that we can no longer afford it.
- Authors must often describe concepts that have not yet been entirely formalized.
- Authors may not know what readers ultimately want.
- Authors may not agree with giving users what they want.
- **If we all agreed upon practices, standards, and definitions, then the value of DAST would be diminished.**

- The **customer** may know what she wants but she may not be able to describe it.
- The **provider** may not have the testing, technology, or domain expertise required for the job.

- **The attitudes:**
- Build the product now. Worry about its infrastructure and process later.
- Build infrastructure first. Worry about the product later.
- A compromise approach (somewhere in between).
- **Testing and quality objectives may not be well defined.**

- **Inability to quantify stability.**
- **Inability to measure stability.**
- **Inability to assess impact of project risks.**

- **Tell me what you want me to do. I'll tell you what I can do, and what it will cost you.**
- **I'll just do it. Then, I'll tell you what's been done.**
- **I'll tell you what I'll do, and then do it.**
- **I'm an expert. So, trust my opinion and let me test.**
- **Customer: I have limited amounts of money to spend.**
 - (1) Tell me what I should do.
 - (2) What can you do for me?
 - (3) I want you to do X.

I'll learn about your product objectives and issues. I'll suggest what should be done. I'll tell you the cost implications, and then I'll begin testing.

1. Define and quantify the tasks (types of tests) that are required.
2. Define and quantify the deliverables.
3. Identify attributes of the information required to do the job.
4. Identify the skill attributes required to do the job.
5. Design and list the *questions* that will be used to collect the pertinent data.
6. Create product documentation, test documentation, test cases, and discover bugs by filling in answers, and by creating more questions.
7. Identify and generalize *questions* that can be reused in the future.

- ❖ **Define and quantify your tasks:**
 - Types of tests to be run.
 - Where each type of test will be run.
 - Number of reruns per type and/or per test area.
 - *The condition variables (configuration and compatibility)*
 - *System requirements*
 - *Supported systems*
- ❖ Number of test cases to be designed and run.

Test Class	Test Effect	Test Characteristic
Bug finding tests	Improve quality	Tests run the first time
QC/Verification tests (regression/acceptance)	Control known quality through comparison	Re-run tests
Measuring tests (milestone acceptance tests, performance tests, benchmark tests, etc.)	Validate/Qualify compliance to set standards	First-time as well as re-run tests

- **Testing the product:**
- Am I responsible for the quality of the testing or the quality of the product?
- What's expected of me?
- What are my deliverables?
- What is the value of my deliverables?
- Do my deliverables meet business objectives?

Step 2: The Deliverables

- **Defining and quantifying your output:**
- Intangibles
 - Delivering products with minimal *quality* risks, under tight *schedule* and *budget* constraints.
- Tangibles (by-products)
 - Documentation, test plans, test cases, bug reports, status reports.
 - The many bugs that you found; and the few bugs that you missed.
 - Number of tests run.
 - Bug report submission rate vs. valid/invalid bug report ratios.
 - Number of testing hours.
 - Good questions and reusable questions.
- **How do you measure your effectiveness for future improvement?**
- Compare current data with historical data.

- **What do I need to start testing?**
 - Information about the product from the business perspective:
 - *Who is the buyer?*
 - *What do we need to do to satisfy the buyer?*
 - *What are the business and financial objectives?*
 - Capture market-share or customer-base?
 - Maximize revenue?
 - Maximize profit?

- **What do I need to start testing?**
- Information about the product from the engineering and deployment perspective:
 - The releases:
 - *Pre-production of a 1st-time release?*
 - *Pre-production of an upgrade/sustain release?*
 - *Post-production release on a new environment?*
 - The changes:
 - *New architecture/infrastructure?*
 - *Coding changes?*
 - *Software environment changes?*
 - *Hardware environment changes?*
 - *Software & Hardware environment changes?*

- **What will I do with the answers I receive?**
 - Depends on what is important to the company/product.
 - Depends on the budget.
 - How much money do I have for testing?
 - How much time do I have for testing?
 - Depends on control.
 - How much control do I have?
 - Depends on the deliverables.
 - Depends on the development life cycle process.
 - How much skill does my development team have?
 - How much skill does my test team have?
 - What affects me?
 - What do I expect of others?

- **Know your job:**
 - How do you measure your output?
 - How do you measure your effectiveness?
- **What affects your output?**
 - Project constraints:
 - Quality attributes
 - Cost
 - Schedule
 - Uncontrollable variables:
 - Process maturity
 - People's capabilities
 - Project stability
 - Changes

- **What affects your effectiveness?**
 - Testing/QA experience
 - Technology experience
 - Domain expertise
- **Identify what you need to do your job.**
- **Strategies:**
 - How do you *improve* controllable variables?
 - How do you *manage* uncontrollable variables?

- **Know the right questions to ask.**
- **Break questions into small units.**
- **Each question should have a specific purpose.**
- **Consider what answers you expect.**
- **Consider what your follow-up questions to certain answers will be.**
- **Consider what the answers will mean to you.**
- **Consider what you'll do with the answers.**
- **Consider follow-up questions for inadequate answers.**
- **Common questions should be organized into checklist and/or fill-in-the-blank format.**
- **Design questions and methods for gathering the required information.**

Step 6: Filling in the Answers

- **Answers can be in multiple choice format.**
- **Answers can be in a “fill-a-blank” format.**
- **Answers can be collected through interviews or the walkthrough process.**
- **It’s okay to fill in the blanks as you go. Know that you will not get all the answers in the first few passes.**

- **Generalize frequently asked questions (FAQs) for reusability.**
- **Group questions into logical groups and simple formats for easy distribution.**
- **FAQs can be used as the basis for product documentation specifications.**
- **FAQs can be used as the basis for training.**
- **FAQs can be used as the basis for test automation.**
- **FAQs can be used as the basis for process improvement.**

- **Testing the Sign In feature**

The image shows a screenshot of a web application's sign-in interface. At the top, there is a dark blue header bar with the text "SIGN-IN" in white. Below the header, the form is enclosed in a light gray border. It contains two input fields: "User Name:" followed by a text box, and "Password:" followed by a text box. Below the password field, there is a small, faint text "(target?)" and a "Sign In" button with a gray background and black border.

An Application Example

Questions to Ask

- How many variables are there on the page?
- Can you list all variables within the page?
- For each variable, can you identify its type?
- Can you list all commands within the page?
- Can you specify the input data associated with each command?
- Can you specify the purpose of each command (e.g., navigation, a request to authenticate sent with input data, a request for a form sent without input data, etc.)?
- For each input variable or command, can you specify the following at the UI, database, and application level?
 - Minimum and maximum length
 - Valid values
 - Default value
 - Error handling mechanism
 - Do you detect invalid conditions? How do you detect an invalid condition?
 - What do you do when an invalid condition is detected?
 - What do you tell the user when there is a failure condition?
- **Is there any business specific logic?**

The image shows a web form titled "SIGN-IN" in a dark blue header. Below the header, there are two input fields: "User Name:" followed by a text box, and "Password:" followed by a text box. Below the password field, there is a link that says "(forgot?)". To the right of the password field is a "Sign In" button.

An Application Example

Questions to Ask Yourself

- **From the user's perspective:**
 - Does the feature set solve user's problems reasonably well?
 - Does the feature set work the way a reasonable user would expect it to work?
 - Are features and error handling implemented with consistency?
- **From the application's perspective:**
 - Does the design and implementation solve the defined problems?
 - Does the design successfully protect the company from quality risks (e.g., functionality, reliability, maintainability, security, etc.)?
- **From the developer's perspective:**
 - Is there anything missing?
 - Could the design implementation be improved?
- **From the tester's perspective:**
 - Where can I find answers to the DAST questions?
 - How can I disprove the validity of the design?
 - How can I find errors in the implementation of the design?



The screenshot shows a web application interface for signing in. At the top left, there is a dark blue header with the text "SIGN-IN" in white. Below the header, there is a white form area. The form contains two input fields: "User Name:" and "Password:". To the left of the "Password:" field, there is a link that says "(forgot?)". To the right of the "Password:" field, there is a "Sign In" button.

Filling in the Answers

								UI		
Control Label	ID/Name	Type	Min. Length	Max. Length	Valid Value	Default Value	Invalid Detection	Invalid Condition Handling	Failure Communication	
Input Data	ID	user_name	HTML Text Box	0	None	All	NULL	None	None	None
	Password	password	HTML Text Box	0	20	All	NULL	(1) Length enforced by the MaxLength property (2) None for value	None	None
Command	Sign In	Sign In	Request with input data	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	Forgot?	N/A	Request without input data	N/A	N/A	N/A	N/A	N/A	N/A	N/A
							Error Handling			

Filling in the Answers

						Application/Business Logics					
	Control Label	ID/Name	Min	Max	Valid Value	App Logic	Invalid Detection	Invalid Condition Handling	Failure Communication	Function	
Input Data	ID	user_name	4	None	[A-Z],[a-z],[0-9],[_]	(1) Minimum 2 numeric and 2 non-numeric characters; (2) No space between characters	Yes	Reject without comparing against database records	Display "Invalid Login" message	Authenticate user	
	Password	password	4	None	[A-Z],[a-z],[0-9],[_]	(1) Minimum 2 numeric and 2 non-numeric characters; (2) No space between characters	Yes	Reject without comparing against database records	Display "Invalid Login" message		
Command	Sign In	Sign In	N/A	N/A	N/A	(1) Compare against ID's and passwords in database (2) Block after 3 invalid logins	Yes	(1) Deny access (2) Stop showing login screen	(1) Display "Invalid Login" message (2) Display "Please contact your administrator" message		
	Forgot?	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	Display a form to allow submission to request for forgotten ID and/or password	
							Error Handling				

Filling in the Answers

	UI										Application/Business Logics							
	Control Label	ID/Name	Type	Min. Length	Max. Length	Valid Value	Default Value	Invalid Detection	Invalid Condition Handling	Failure Communication	Min	Max	Valid Value	App Logic	Invalid Detection	Invalid Condition Handling	Failure Communication	Function
Input Data	ID	user_name	HTML Text Box	0	None	All	NULL	None	None	None	4	None	[A-Z],[a-z],[0-9],[_]	(1) Minimum 2 numeric and 2 non-numeric characters; (2) No space between characters	Yes	Reject without comparing against database records	Display "Invalid Login" message	Authenticate user
	Password	password	HTML Text Box	0	20	All	NULL	(1) Length enforced by the MaxLength property (2) None for value	None	None	4	None	[A-Z],[a-z],[0-9],[_]	(1) Minimum 2 numeric and 2 non-numeric characters; (2) No space between characters	Yes	Reject without comparing against database records	Display "Invalid Login" message	
Command	Sign In	Sign In	Request with input data	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	(1) Compare against ID's and passwords in database (2) Block after 3 invalid logins	Yes	(1) Deny access (2) Stop showing login screen	(1) Display "Invalid Login" message (2) Display "Please contact your administrator" message	
	Forgot?	N/A	Request without input data	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	Display a form to allow submission to request for forgotten ID and/or password
Error Handling										Error Handling								



Potential buffer-overflow error

■ **Project Goal:**

*(Looking for: Why are we making the product?
How will testing help accomplish this goal?)*

- What is the primary business objective for this test project?
- What is the purpose of testing this application or system?
- What are the selling points of the product?
- How is our product different from its competitors?
- Is the competitor releasing a product during the same timeframe?

- **Documentation:**
(*Looking for:* Is there any documentation? What will be available and when? Who is writing and updating the documentation?)
 - **Product** (*Looking for:* ReadMe, Manual, Demo, Tutorials, Box Copy, Advertisements)
 - What Product documents exist now?
 - What is planned for this product?
- **Project** (*Looking for:* Product Proposal, Product Design, Engineering Specifications, Marketing/Business Requirements)
 - What project documents exist now?
 - When will the project documents be completed?
 - Who is responsible for editing/updating the project documents?
 - Is there a change request system in place?
 - Does a BOM exist now?
- **Test** (*Looking for:* How much does the team know about testing? How much do they care about the testing project or process?)
 - What testing documentation is expected?
 - Will the development team review and approve a test plan?
 - Will the development team need to see test cases?
 - Does the development team want test status reports?

❖ Schedule

- Product Release (*Looking for: Is the proposed schedule realistic?*)
- When is the proposed product ship date?
- Are the manufacturing dates booked?
- Is there room for negotiation?
- Is the Product launch coordinated with other events?
(Conference? Retail season? Marketing campaign? Super Bowl?)

❖ Development Schedule (*Looking for: How well has the schedule been thought through? Is there room for negotiation in milestones?*)

- Are dates set for UI Freeze? Alpha? Beta? Etc.
- When will we receive testable code?
- What milestone has the product currently reached?
- Is there a feature implementation schedule?
- When is the project kick-off meeting?

❖ Test Schedule

- When will we start testing?
- How much time is allotted for testing?
- When is testing expected to be completed?

- **Practice software-testing “archeology”**
The systematic recovery and study of material evidence.
- **Be a leader**
Take the time to explain what you do and why.
Explain the risks of not doing certain things.
- **Be an educator**
Teach others what you know.
- **Be a student**
Learn from others what you don't already know.

About Hung Q. Nguyen

Hung Q. Nguyen is Founder, President and CEO of *LogiGear Corporation*, a Silicon Valley software testing company whose mission is to help software development organizations deliver the highest quality products possible while juggling limited resources and schedule constraints. *LogiGear* offers many value-added services including application testing, automated testing and web load/performance testing for e-business and consumer applications. Nguyen's company produces and markets TRACKGEAR™, a web-based defect tracking system. *LogiGear* also specializes in Web application, hand-held communication device and consumer electronic product testing, and offers the software development community a

comprehensive “Practical Software Testing Training Series.” In the past two decades, Nguyen has held leadership roles in business development, engineering, quality assurance, testing, product development, and information technology. Nguyen is the author of *Testing Applications on the Web* (Wiley) and co-author of the best-selling book, *Testing Computer Software* (Wiley). He also develops and teaches software testing courses for UC Berkeley and UC Santa Cruz Extension, and for LogiGear. He holds a Bachelor of Science in Quality Assurance from Cogswell Polytechnical College, and is an ASQ-Certified Quality Engineer and active senior member of American Society for Quality.

About Michael Hackett

Michael Hackett, Vice President, Business Strategy and Operations, is a founding partner of *LogiGear* Corporation. He has over a decade of experience in software engineering and the testing of shrink-wrap and Internet based applications. Michael has helped well-known companies including Palm Computing, Oracle, CNET, Electronics for Imaging, Adobe Systems, The Learning Company, Power Up Software, PC World and The Well produce, test and release applications ranging from business productivity to educational multimedia titles — in English as well as a multitude of other languages. Michael holds a Bachelor of Science in Engineering from Carnegie-Mellon University.

LogiGear Corporation provides testing expertise and resources to software development organizations. Our partners benefit from our seasoned testing staff and facilities, practical training programs, and test support products. We help development teams deliver high quality software, improve time-to-market, and optimize development productivity. Founded in 1994, *LogiGear* has built a reputation on partnering with software development organizations to help make the most of outsourcing and staff training solutions. We assist our clients in delivering the best possible quality products while juggling limited resources and schedule constraints.

