Software Testing & Test Automation

National Defense Industrial Association
Software Test & Evaluation Summit/Workshop
Washington, DC

Hung Q. Nguyen
LogiGear® Corporation
September 15, 2009

© 2009 LogiGear Corporation. All Rights Reserved
Objectives

• Offer a briefing of current software testing state-of-the-practice and trends in the business world

• Outline the needs and challenges of test automation, particularly when dealing with high volume and high rate-of-change; and approaches to dealing with these challenges

• Offer recommendations on strategy, selection and acquisition of software test programs
Agenda

• The evolution of software testing
• Software testing today—the shift in direction
• A test strategy framework to consider
• Test automation challenges
• A brief introduction of test automation approaches—contrasting methods, technologies, people and processes
• Recommendations
Early Stage

- Software testing was not cool!
- Testing was an entry to advancing into a better and brighter non-testing career.
- “Testing was for unskilled people?”
- Testing tools were expensive, hard-to-use, and did not address productivity.
- Business was mostly disconnected from testing.
Post Dot.Com Stage

• Testing is cool!
• Everybody tests...
• “Testing is for technical people with coding skills?”
• Certification widely marketed to non-developer testing people...
• Tools are everywhere… (more and more are Open Source).
• Testing is moving offshore
• Business is still mostly disconnected from testing.
Software Testing Today

• Testing is still cool, and everyone should do it!
• Build strategic, end-to-end framework for effective handling of test automation and software high rate-of-change
• Clear budget allocated to build quality management infrastructure as well as test production
• Transparency (visibility) is key
  – Business, from executives to the entire product team, is now very connected with what goes on in testing
  – Focus on building trust and confidence
What Business People Think

“Software testing is complex, but I am tired of finding bug after bug that a 5th grader wouldn’t have turned in. Virtually, every technical product today includes a lot of software. It’s a rare engineer that can nearly write perfect code. **Methodical and thorough testing of software is the key to quality products that do what users expect.**”

Steve Wozniak, CTO, Wheel of Zeus, Apple Computer co-founder

“A great majority of software tests are designed to **confirm the existence of features, instead of attempting to break the software.**”

Sue Kunz, CEO of Solidware Technologies

"**In theory, test automation** is supposed to be a silver bullet to increase test coverage and improve quality; **offshoring is supposed to drastically cut costs.** In reality, many organizations struggle with both, and don’t see significant gains despite extensive efforts.”

Bruce Martin, Vice President, Product Strategy, PSS Systems

"**Automation isn't just about technical decisions.**"

Michael Hatam, President, Application Services, Moyo Group

“Software testing is surprisingly low-tech, and requires too much ‘hands-on.’ **Would most people run their virus software frequently if they had to invoke it manually?**”

Sue Kunz, CEO of Solidware Technologies

"**Offshoring by itself is not enough. You need testing strategies to stay ahead of the competition and maximize your investment.**"

Robert S. Alvin, Chief Executive Officer and Chairman, Netline Corporation

"**Despite continued advances in development techniques and technologies, software quality problems are as pervasive as ever.** Software testing teams are under tremendous pressure to test more complex systems with the same or fewer resources, and corporate managers are always looking to shave costs by leveraging offshore testing.”

Adam Au, Vice President, Engineering, Centrify Corporation

Source: Global Software Test Automation, 2006, HappyAbout, Nguyen et. al
Software Testing Today

• Testing as a whole, not in parts
• Team-based solution
• Clear leadership or ownership.
• Ask What, When, Who, How Much, and **Why**, not How
• Build lasting asset
  – Code -> Cool feature -> (If works) Increase Revenue
  – Test -> Well-behaved Code/Feature  -> Increase Revenue
  – **Code + Test** -> **Asset (Tested Code)**
Software Testing Today

• Targeting SP³ = People + Practice + Process
  – Practice = Method + Tool
    • Method first
    • Test design, not programming focused

• Know your “…ility”s
  – Feature—reliability, compatibility (backward), usability, interoperability
  – Performance—acceptability (response time), availability, scalability
  – Security
  – Compliance—regulatory such as accessibility, privacy, traceability, etc.
<table>
<thead>
<tr>
<th>Culture, Methodology, Belief, Technique &amp; Other Changes Involved</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Agile versus waterfall</td>
</tr>
<tr>
<td>• Test-driven development versus traditional software testing</td>
</tr>
<tr>
<td>• Developer testing versus system testing</td>
</tr>
<tr>
<td>• Technical versus non-technical testing staff</td>
</tr>
<tr>
<td>• Source-based testing versus integration/system testing</td>
</tr>
<tr>
<td>• Context-driven versus non-context-driven</td>
</tr>
<tr>
<td>• Bug-hunting versus verification/validation</td>
</tr>
<tr>
<td>• COTS versus Open Source downloads versus build</td>
</tr>
<tr>
<td>• Automation versus manual testing</td>
</tr>
<tr>
<td>• Exploratory versus scripted testing</td>
</tr>
<tr>
<td>• Certification versus skill and education</td>
</tr>
<tr>
<td>• RUP versus V-Model</td>
</tr>
<tr>
<td>• Outsourced versus insourced</td>
</tr>
<tr>
<td>• Offshore versus onshore</td>
</tr>
<tr>
<td>• Light process versus CMM</td>
</tr>
<tr>
<td>• Etc.</td>
</tr>
</tbody>
</table>
## Customer Satisfaction is Key!

<table>
<thead>
<tr>
<th>THINGS OF THE PAST</th>
<th>SOFTWARE TESTING TODAY</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OPTIMIZING</strong></td>
<td></td>
</tr>
<tr>
<td>Process</td>
<td>Productivity (Bugs and Tests)</td>
</tr>
<tr>
<td><strong>TARGETING</strong></td>
<td></td>
</tr>
<tr>
<td>Process Maturity</td>
<td>(SP^3 = \text{People + Practice (w/ Automation)} + \text{Process})</td>
</tr>
<tr>
<td><strong>DRIVING</strong></td>
<td></td>
</tr>
<tr>
<td>CMM, ISO, TQM</td>
<td>Efficiency</td>
</tr>
<tr>
<td>Test things right</td>
<td>Test the right things right</td>
</tr>
<tr>
<td>Certification</td>
<td>Skill and result</td>
</tr>
<tr>
<td>Business is disconnected</td>
<td>Business is connected and demands transparency</td>
</tr>
<tr>
<td>Quality</td>
<td>Quality with proof</td>
</tr>
<tr>
<td>QA</td>
<td>Test to seek out bugs</td>
</tr>
<tr>
<td>QC</td>
<td>Built-in quality and trust</td>
</tr>
<tr>
<td>Test Procedure</td>
<td>Test Automation</td>
</tr>
</tbody>
</table>
Test Strategy Framework

- Determine Quality Objectives
- Select Test Approaches
- Select Test Types
  - Bug Types
  - Targeted Interface
  - Application Stability Requirements
- Determine Execution Test Phases
- Apply Test Design Methods/Techniques
- Select Test Tools

Write Test Plan

Write Test Cases
• The test strategy describes your planned approaches to testing an application with the focus on asking why one approach is better than the other, instead of asking how. Normally, there are two main goals of testing: Bug finding and/or validation. Once you have settled upon the goals for the test project, you can strategize on how you will go about accomplishing those goals.

• Quality objective is a concrete requirement term that specifies the type of the quality objective and the level of acceptability for each.

• Test approach is a testing philosophy or style that drives the selection of certain test design methods or techniques.

• Test type is a category of test activities with the objective of exposing specific types of bugs at certain interface, applicable for various level of product maturity. The selection of the test types must satisfy the test or quality objectives.

• Bug type—bugs or software errors are often grouped into categories or types so we can condense a huge list of issues into a conceptually manageable group of ideas. Any approach to categorizing bugs is arbitrary.

• Interface—a boundary across which, two independent systems meet and act on or communicate with each other. In computer technology, there are several types of interfaces.

• Application stability—The state or quality of being stable of a software application. In software development, a stable application or functionality in an application is one that is reliable and dependable, and will consistently produce the same expected outputs given on the same set of inputs and conditions.

• A test phase is a time block often derived from the SDLC phases. Within each SDLC phase, there might be more than one test phase. Each test phase has different objectives related to bug finding or validation, described in each section. Testing staff participates in ongoing project management activities.

• A test design method is a systematic procedure in which you create tests to be executed. A test type or test approach may use one or more test design methods.
Test Automation

- Testing done by development versus done by testing/QA staff
- Test automation of
  - Load performance
  - Security
  - Feature (functional/integration)

This discussion focuses on functional/integration testing where testing is challenged by large volume of test cases, test data, user scenarios, permutation of operating configurations and environments, and high rate-of-change.
Test automation provides great benefits to the software testing process and improves the quality of the results.

- It improves reliability while minimizing variability in the results
- Speeds up the testing and increases test coverage
- Make it possible to test rapid releases
- Use machine to run mundane, routine tests, and rerun large volume of tests, enabling valuable human test resource to focus on breaking the software and seeking out new bugs
- Ultimately can provide greater confidence in the quality of the software being tested
Automation Challenges

- Too much programming
- High rate-of-change (software)—high cost of maintenance
- Low reusability/scalability
- Too many platforms/technologies
- Lack of advanced output/object validation capability
- Too expensive if the returned value is low
Automation Challenges

• Business consideration
  – The need of high-volume test automation
  – The cost of ownership (Assuming it is a necessity or a profitable decision to automate)
    • Technology/Tool Licensing and/or Development Cost
    • Production Cost
      1. One-time cost to design, implement and execute a test
      2. Recurring cost of troubleshooting test execution failures including false-negatives
      3. Recurring cost of updating and validating execution of existing tests (due to changes)
• Business consideration
  – The Cost of Owning an Automated Test

Cost of an Automated Test* = \frac{(Technology Cost + Production Cost)}{Volume (#) of Tests}

* All test cases are not equal—a clear and structured definition of a test case is required. Furthermore, this cost will be a moving target over time.
Automation Challenges

• Technology
  – Multiplatform (cross-) support
    • Ability to interface with multiple platforms
    • Ease of extensibility to new platforms
    • Reusability of test suites/scripts across multiple platforms
  – Output/Object validation support
Automation Challenges

• Method
  – Maintainability—handling high rate-of-change
  – Reusability—ability to reuse common test components without having to program
  – Scalability—ability to automate high volume of tests
  – Visibility
    • Tests auditable by management and non-programmer staff
    • Productivity is structurally measurable
Outflows of Test Automation

Visibility -> Control -> Manageability
  Measurability

Reusability -> Productivity
  - By quantity
  - By quality

Scalability

Maintainability
  Cost Efficiency (Minimize Costs of Ownership)

Tangible Benefits
  - Improved time-to-market
  - Improved quality of releases
  - Improved predictability
  - Improve communication
  - Higher test coverage
  - Lower testing costs
  - Test early. Test Often
  - Lower support costs
  - Effective use of resources
  - Improved customer confidence & adoption
1. Tests are treated as product asset, along with the source code.
2. Tests, good or bad are dependent of the design.
3. Tests, manual or automated must be optimized for visibility, reusability, scalability and maintainability.
4. Tests must be automation-ready.
5. Tests, if they are worth automating, should follow the 5% rule:
   - No more than 5% of all tests should be executed manually
   - No more than 5% of all efforts around testing should involve automating the tests
   - No more than 5% of coded test scripts against non-coded test scripts.
A Brief Introduction of Test Automation

- Record/Playback
- Scripting approach
- Data-driven scripting approach
- Action keyword approach
Record/Playback

- One time recording of actions and checks
- Multiple time playback

Test Tool
(Invisible to the target system)

Recorded test scripts

- Actions
- Checks

Log

Test data entered by testers or users

Target System/AUT

Actions
Checks
Results
EXAMPLE

A Recorded Script

select window "Logon"
enter text "username", "administrator"
enter text "password", "testonly"
push button "Ok"
select window "Main"
push button "New Customer"
expect window "Customer Information"
select field "Name"
type "Jones"
select field "First Name"
type "John"
select field "Address"
type "54321 Space Drive"
Scripting Approach

• Automation is regarded as a programming activity
• Parameterize hard-coded values
• Separate data from code by moving variables to include files
• Create utility functions to be shared
• Produce and maintain like any other software
• Train black-box test engineers to run the scripts
Function EnterCustomerJones
  Logon
  Press "New Customer"
  Enter Field "Name", "Jones"
  Enter Field "First Name", "John"
  Enter Field "Address", "54321 Space Drive"
  ...
  Logoff
End Function

Function OrderProduct
  Logon
  Press "New Order"
  Enter Field "Product", "OurProduct 1.0"
  Enter Field "Amount", "35"
  Enter Field "Delivery", "asap"
  ...
  LogOff
End Function

...
(Scripting) Data-Driven Approach

• Scripts are parameterized and driven using a series of arguments
• Take advantage of tester’s familiarity with test case design and creation using tables and matrices
• Accommodate localization projects
• Recognize the importance of patterns in test cases
• Enable testers to catalog test cases with spreadsheets in software such as Excel
• Enable testers to specify expected results in spreadsheets
GOODMORNING

goodmorning

GoodMorning

Input Data
(1) File with test data:

<table>
<thead>
<tr>
<th>nr</th>
<th>case</th>
<th>whole</th>
<th>pattern</th>
<th>matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>off</td>
<td>off</td>
<td>GOODMORNING</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>off</td>
<td>on</td>
<td>GOODMORNING</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>on</td>
<td>off</td>
<td>GOODMORNING</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>on</td>
<td>on</td>
<td>GOODMORNING</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>off</td>
<td>off</td>
<td>goodmorning</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>off</td>
<td>on</td>
<td>goodmorning</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>on</td>
<td>off</td>
<td>goodmorning</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>on</td>
<td>on</td>
<td>goodmorning</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>off</td>
<td>off</td>
<td>GoodMorning</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>off</td>
<td>on</td>
<td>GoodMorning</td>
<td>3</td>
</tr>
<tr>
<td>11</td>
<td>on</td>
<td>off</td>
<td>GoodMorning</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>on</td>
<td>on</td>
<td>GoodMorning</td>
<td>1</td>
</tr>
</tbody>
</table>

(2) Text file used in the test:

goodmorning
GOODMORNING
GoodMorning
GoodMorningVietnam

(3) Data driven script:

for each line in the file do
    open the find dialog
    read a line from the file
    use the values to fill dialog
    press find button
    check amount of matches
    close the dialog
• Separation of action, interface, logic (the steps in the scripts) and data
  – Action can also go in a separate file or spreadsheet
  – Interface can go in a separate file or spreadsheet
  – Test logic can go in a separate file or spreadsheet
  – Test data can go in a separate file or spreadsheet
• Action can be reused
How Does it Work?

- **Test Cases** are written or described through a series of **Test Lines** using parameterized and reusable **Action** keywords

<table>
<thead>
<tr>
<th>Action keyword</th>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add product</td>
<td>12345678</td>
<td>Hammer</td>
<td>For pounding nails</td>
</tr>
</tbody>
</table>

- **Actions** are created and communicate with target systems and software applications including the application under test (AUT) through one or more supported **Interfaces** to carry out intended tasks

  Note:
  
  For now, think of an **Action** as a verb or something that you do

- Information about **Interfaces** such as Windows, Web or Java graphical user interface (GUI) or command-line interface (CLM) are later defined to enable actions to send inputs to and receive outputs from the target system and/or AUT
How Does it Work?

- **Test Case** logic and information along with other test case related information is stored in a **Test Module**
- **Action** (high-level, application) information is stored in an **Action Definition Worksheet** (low-level, system actions are built-in)
- **Test Data Set** (for data-driven testing) is stored in a separate **Data Set Worksheet**
- **Interface** information is stored in an **Interface Definition Worksheet**

See an example
An Application Under Test

This GUI application contains three dialog boxes:

- **Main** dialog box
- **Add A New Item** dialog box
- **Update Product Information** dialog box
A test case with 6 test lines using 5 actions.

Test Module

Parameters in an action

Action Definition Worksheet

Interfaces used in actions
### Action Definition Worksheet

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTION DEFINITION</td>
<td>add product</td>
<td></td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>default value</td>
<td>description</td>
<td></td>
</tr>
<tr>
<td>argument</td>
<td>number</td>
<td>name</td>
<td>description</td>
</tr>
<tr>
<td>use interface</td>
<td>inventory</td>
<td>mode</td>
<td>windows</td>
</tr>
<tr>
<td>click</td>
<td>source</td>
<td>main</td>
<td>control</td>
</tr>
<tr>
<td>enter values</td>
<td>window</td>
<td>add product</td>
<td>#number</td>
</tr>
<tr>
<td>click</td>
<td>source</td>
<td>add product</td>
<td>control</td>
</tr>
</tbody>
</table>

### Interface Definition Worksheet

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERFACE ENTITY</td>
<td>Main</td>
<td></td>
<td></td>
</tr>
<tr>
<td>interface entity setting</td>
<td>title</td>
<td>ABT Inv</td>
<td></td>
</tr>
<tr>
<td>interface element</td>
<td>ta name</td>
<td>ta class</td>
<td>caption</td>
</tr>
<tr>
<td>interface element</td>
<td>add</td>
<td>button</td>
<td>Add An Item</td>
</tr>
<tr>
<td>interface element</td>
<td>update</td>
<td>button</td>
<td>Update An Item</td>
</tr>
<tr>
<td>interface element</td>
<td>end</td>
<td>button</td>
<td>End</td>
</tr>
<tr>
<td>interface element</td>
<td>ta name</td>
<td>ta class</td>
<td>caption</td>
</tr>
<tr>
<td>interface element</td>
<td>auto</td>
<td>checkbox</td>
<td>Check1</td>
</tr>
<tr>
<td>interface element</td>
<td>products</td>
<td>class</td>
<td>local pos</td>
</tr>
<tr>
<td>interface element</td>
<td>products</td>
<td>combobox</td>
<td>combobox1</td>
</tr>
<tr>
<td>interface frame</td>
<td>level</td>
<td>product information</td>
<td>caption</td>
</tr>
<tr>
<td>interface frame</td>
<td>1</td>
<td>Product Information</td>
<td></td>
</tr>
<tr>
<td>interface element</td>
<td>ta name</td>
<td>ta class</td>
<td>local pos</td>
</tr>
<tr>
<td>interface element</td>
<td>number</td>
<td>textbox</td>
<td>textbox1</td>
</tr>
<tr>
<td>interface element</td>
<td>name</td>
<td>textbox</td>
<td>textbox2</td>
</tr>
<tr>
<td>interface element</td>
<td>description</td>
<td>textbox</td>
<td>textbox3</td>
</tr>
<tr>
<td>interface element</td>
<td>quantity</td>
<td>textbox</td>
<td>textbox4</td>
</tr>
</tbody>
</table>

**Interfaces used in actions**
<table>
<thead>
<tr>
<th>Approach</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record &amp; Playback</td>
<td>A useful way to learn about the testing tool and automation concept</td>
<td>• Not maintainable b/c it is highly sensitive to changes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• GUI support only</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Test script readability is very low</td>
</tr>
<tr>
<td>Scripting</td>
<td>• Improved reusability of utility scripts</td>
<td>• Test script readability is still low, which makes it difficult to manage the project</td>
</tr>
<tr>
<td></td>
<td>• Improved maintainability</td>
<td>• Low maintainability b/c it is still sensitive to changes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Test creation is done by programmer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• One-to-one test-case-to-test-script</td>
</tr>
<tr>
<td>Data-driven</td>
<td>Improved reusability and maintainability</td>
<td>• Business logic (steps) are still in automated scripts</td>
</tr>
<tr>
<td></td>
<td>• Separation of action and data</td>
<td>• Maintainability is poor for business logic changes</td>
</tr>
<tr>
<td></td>
<td>• Test data can be stored in table/spreadsheet file</td>
<td>• Test script readability is still low, which makes it difficult to manage the project</td>
</tr>
<tr>
<td>Keyword-driven</td>
<td>• Highly maintainable—separation of action, interface, logic (the steps in the scripts) and data drastically lower the cost of maintenance in handle high rate-of-change</td>
<td>• A well designed framework must be in place</td>
</tr>
<tr>
<td></td>
<td>• Highly reusable—commonly used modular, reusable test components are built by test architects/automation engineers, then assembled into test scripts by test designers.</td>
<td>• Staff needs to be trained in test design</td>
</tr>
<tr>
<td></td>
<td>• Highly scalable—transaction-level modular test components can be built/extended by using several existing modular test components</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Highly transparent—tests auditable by management and non-programmer staff.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Productivity is structurally measurable</td>
<td></td>
</tr>
</tbody>
</table>
Scalability Illustration

- Tests: 3000, 6000
- Application (High-level) Action Keywords: 200, 250
- Built-in/System (Low-level) Action Keywords: 20, 22
# Division of Work

<table>
<thead>
<tr>
<th>Approach</th>
<th>Automation Engineering</th>
<th>Test Case Production</th>
<th>Test Data Construction / Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record &amp; Playback</td>
<td>N/A</td>
<td>Test Engineer</td>
<td>N/A</td>
</tr>
<tr>
<td>Scripting</td>
<td>Automation Engineer</td>
<td>Automation Engineer</td>
<td>N/A</td>
</tr>
<tr>
<td>Data-driven</td>
<td>Automation Engineer</td>
<td>Automation Engineer</td>
<td>Test Engineer</td>
</tr>
<tr>
<td>Keyword-driven</td>
<td>Automation Engineer</td>
<td>Test Engineer</td>
<td>Test Engineer</td>
</tr>
</tbody>
</table>

Note: Programming skill is required for Automation Engineer
Tool Options

<table>
<thead>
<tr>
<th></th>
<th>Visibility</th>
<th>Reusability</th>
<th>Scalability</th>
<th>Maintainability</th>
<th>Code-free</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action-Based Testing</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Record-Play Back</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>2.0  Approach</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Data-Driven Approach</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Scripted Approach</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Record-Play Back</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

Legend
0: Very Poor
1: Marginal
2: Good
3: Very Good
Recommendations

- Develop a testing strategy first—know your “…ility”s
- Consider using a framework to select the test approaches, types, design techniques and supporting tools that work best for your program
- You need process but process itself does not solve your testing problem—focus on your people + practice + process (SP³)
- Test automation—understand your cost-of-ownership well
- Achieving high volume automation is very challenging, but you need to get there—pay attention to maintainability (handling high rate-of-change), reusability, scalability and maintainability
- Treat your tests as assets
- In evaluating automation technologies, consider multiplatform (cross-) support—ability to interface with multiple platforms; ease of extensibility to new platforms; reusability of test suites/scripts across multiple platforms; and output/object validation support
- To achieve high-volume automation, method comes first, then tool! Your automation must be test design-centric, not programming-centric
LogiGear is a world leader in high-volume software test automation production, serving hundreds of companies, from Fortune 500 to startups; draw from 15+ years of research and practice, 5 publications, including one bestseller, 18 pragmatic training courses, a proven automation methodology and a software framework that elevates automation test coverage to exceed 95% on a diverse set of technology platforms, and 400+ skilled engineers in the US and Vietnam, it delivers unbeatable value that precisely meets each customer’s specific needs.

www.LOGIGEAR.com

/GSA Contract—Schedule 70 # (In process)/