DEDICATED TO SHOWCASING NEW TECHNOLOGY AND WORLD LEADERS IN SOFTWARE TESTING

LogiGear MAGAZINE

THE AGILE TEST AUTOMATION ISSUE

Action Based Testing: The Solution for Agile Test Automation

By Hans Buwalda, CTO LogiGear

RELATED ARTICLES

Automation Tools in Full View
By Bogdan Bereza-Jarocinski

Do Testers Have To Write Code?
By Elisabeth Hendrickson

SPOTLIGHT INTERVIEW

Jonathan Rasmusson, Author of *The Agile Warrior*

March/April 2011 | Volume V | Issue 3

EDITOR'S LETTER

I have been excited about this issue since I included it in the 2011 editorial calendar. This issue of *LogiGear Magazine* dives into an exploration of agile automation—from the most efficient methods for test automation, to skill sets and better preparation for test teams, and even to understanding the variety of tools in question. We are preparing you for the discussions and implementations to successfully automate agile projects.

The problem of automating agile projects revolves around the agile *need for speed* paired with lean, light-weight documentation—ideals withholding the keys to successful automation from the past: time and information. How can test teams be agile, fast, and lean, yet still automate more in less time with less prep time and less information?

To address the challenges and fears of implementing automation in agile projects, LogiGear CTO Hans Buwalda presents Action Based Testing (ABT) as the best solution for test automation in the new dynamics of agile projects; Bogdan Bereza-Jarocinski provides an overview of test tools followed by Elisabeth Hendrickson's expert thought into whether or not testers must now write code; "Spotlight Interview" with Jonathan Rasmusson, author of *The Agile Warrior*.

This issue includes our continuing glimpse into Viet Nam through a Q&A session with McAfee representatives who visited for a week ,and an analysis report from the LogiGear 2010 Global Testing Survey focusing on the *real world* of software testing directly from practitioners.

I hope you enjoy this issue on automating in agile projects and can refer back to it for your current and future agile endeavors!

Michael Hackett
Senior Vice President
Editor In Chief

IN THIS ISSUE

IN BRIEF

FIVE-PART ARTICLE SERIES ON AGILE TESTING AVAILABLE ONLINE

LogiGear Senior Vice President Michael Hackett provides an overview of agile testing categorized into five sections:

Part I: Introduction to Agile
Part II: Agile is About People
Part III: Practices and Processes
Part IV: Skills and Training
Part V: Tools

LOGIGEAR CHANNEL

Michael Hackett: Agile Automation

http://www.logigear.com/webcast/

Michael Hackett: Agile Testing: New Roles for Traditional Testing in Agile (Parts 1-6)

http://www.logigear.com/resource-center/agile-resource-center/agile-webcasts.html?layout=webcast

Hans Buwalda: Agile Test Development
http://www.logigear.com/march-issue-2011/1029-hans-buwalda-agile-test-development-.html

LOGIGEAR OPENS NEW CENTER IN DA NANG

Da Nang City, March 12, 2011- Investing in human resource growth throughout Viet Nam is one of LogiGear's objectives in order to expand its leadership position in outsourced software testing. In anticipation of an explosive business growth and global demand for qualified software testers in Viet Nam, the new facility is expected to employ over a hundred engineers recruited and trained within Da Nang by the end of 2011. It is LogiGear's goal to develop the Viet Nam software testing community and place the country as the industry's destination of choice for outsourcing.

Blogger of the Month

Tim Ottinger

Book author and avid blogger, Tim Ottinger has been in the software industry for over thirty years. Under his domain, Ottinger has two blogs: *Agile Otter* and *Agile in a Flash*. In the latter, Ottinger and his co-author Jeff Langr keep readers posted on their techniques and methods in agile testing. Here is an excerpt from *Agile in a Flash*:

"There are many ways to conduct an agile project. Some work with huge backlogs, some with spur-of-the moment requirements, some have continual release, some have non-time-boxed continual flow. We recommend starting with the structure shown on this Agile in a Flash card (http://agileinaflash.blogspot.com/2010/08/basic-agile-flow.html).

Following this plan, the customer team puts together a prioritized list (feature backlog) of desired features for the upcoming product release.

The release is broken into iterations and the team and customer agree on what will be delivered at the start of each iteration (no sooner). The iteration is of fixed length, something that allows the team to begin gathering consistent data, which in turn they feed back into their estimates and subsequently a larger plan. Upon incrementing and iterating the software a few times, the software reaches a state that it may be released to the customer. Does the system implement the agreed-upon features (did it pass all of its acceptance tests)? Yes: Release to production!

The flow outlined above is a reasonable starting point for a team transitioning to agile. It represents a kind of "Shu" in the Shu-Ha-Ri cycle, where one follows a certain technique or style for a while to build up their ability to perform it. In fact, both of us started with this basic pattern and found that it worked just fine. As you move to more "Ha" stage, you might experiment with reducing the size of stories so that more of them are done and "in the can" before the end of the iteration. . . . "

Read more at http://agileinaflash.blogspot.com/ or http://agileotter.blogspot.com/.■

FEATURE ARTICLE

Action Based Testing: The Solution for Agile Test Automation

LogiGear CTO Hans Buwalda summarizes ABT's advantages for agile projects with focus on the relationship between product and automation development life cycles, and the importance of test modules in test design.

How can automated functional testing fit into agile projects? That is a question we encounter a lot nowadays. Agile has become more common, but functional testing often remains a manual process because during agile iterations/sprints, there is simply not enough time to automate it. This is unlike unit testing, which is routinely automated effectively. The short answer is:

1. A well planned and organized test design and automation architecture
2. Organize the test design and automation architecture into separate life cycles

In this article I will show how the Action Based Testing method can help you to do just that. Let me first introduce Action Based Testing, followed by discussing how it can make both test design and test automation fit the demands of agile projects.

Action Based Testing

There are various sources where you can read more about Action Based Testing. Let me summarize the key principles here that are at the core of the method:

1. Not one, but three life cycles

It is common to have testing and automation activities positioned as part of a system development life cycle, regardless of whether that is a waterfall or an agile approach. ABT however regards three distinct life cycles. Even though they have dependencies on each other, in an ABT project they will be planned and managed as separate entities:

1. **System Development**: follows any SDLC, traditional or agile model.
2. **Test Development**: includes test design, test execution, test result follow up, and test maintenance.
3. **Automation**: focuses solely on the action keywords, interpreting actions, matching user or non-user interfaces, researching technology challenges, etc.

2. Test Design

The most important property is the position of test design. It is seen as the single most enabling factor for automation success, much more than the actual automation technology. In ABT, it is considered crucial to have a good "high level test design" in

which so called "test modules" are defined. Each test module should have a clear scope that is different from the other and is developed as a separate "mini project."

A test module will consist of test objectives and action lines. The test objectives outline the scope of the test module into individual verbal statements defining what needs to be tested in the module.

The tests in the test module (which looks like a spreadsheet) are defined by a series of "action lines," often further organized in one or more test cases. Every action line defines an "action" and consists of an "action word" defining the action, and arguments defining the data for the action, including input values and expected results.

Note here the ABT test case figures, not as central as in some other methods. We feel the test case is too small and too isolated of a unit to give good direction to test development. Rather than having a predefined list of test cases to be developed, we like to make a list of test modules, and let the test cases in them be the result of test design, not the input of it.

Consequences derive from varying test cases and increase significantly during the creative process. Also, each test case can leave behind the preconditions of the next, resulting in a good flow of the test execution.

3. Automation

In ABT the automation activity is separated from the test development. Test design and automation require very different skill sets and interests. There might be people that are interested at doing both, which is fine, but in my experience that is not very common. Also it assigns ownership for "getting the test to work."

In ABT the automation engineers will concentrate on automation of actions and making "interface definitions" to manage the interaction with the interfaces (user or non-user) of the system under test. This type of automation activity requires advanced skills and experience.

Agile Test Development

In using ABT with its separate life cycles for test development and test automation, there are in fact two topics addressing how to fit automated testing in agile projects:

1. Test design and development
2. Automation

As explained earlier, I see testing and test automation entitled to its own life cycles in addition to the system development life cycle. Apart from how agile the main project is, testing and test automation progress individually.

Having said that and using a scrum project with sprints, testing activities in an agile project fall into three timelines:

1. Testing in regular system development sprints
2. Test development prior to development sprints
3. Testing after development has finished

1. Testing in regular sprints

The most common practice is, and will remain, to develop and execute tests as part of sprints. In a sprint, functionality is progressively understood from user stories and conversations to become clear enough for testers to test it. This can be done in developed tests similar to ABT test modules, as well as exploratory and interactive testing. It can also be good practice to

capture at least some of the "interesting" interactive tests in test modules for future use.

Unit tests are an invaluable asset, but in the ABT approach one would like to consider options to re-use and extend their reach across the lines of addressing single functions.

By defining test modules for unit tests and assigning them to actions, they can be strung together more easily to test with a wider variety of values and include other parts of the system under test, either during a sprint or later on.

2. Test development prior to development sprints

In the ABT method the use of actions, in particular high business level actions, allow for the development of tests with a non-technical focus on business functionality, often simply called "high level tests." Such tests stay away from details in the UI and focus on business transactions, like requesting a home loan, or renting a car.

Higher level tests can be developed early in a project. These tests don't have to wait for a system development sprint since there will be limited time to carefully understand business functionalities and create appropriate tests for them.

The number of, and whether or not business level tests can be made, depends on individual situations. In general, I would recommend the following:

- Have as many business level tests as possible, as they add great value to overall depth and quality, as well as being resilient against system changes that do not pertain to them.
- Use the high level test design step in ABT (where the test modules are identified) to determine what can be done early on in business level tests, and what needs to be completed in detail tests as part of development sprints.

3. Testing after sprints

Once sprints for individual system parts have finished and these parts come together, normally more testing will be needed to ensure quality and compliance of the entire system. Also, tests may be needed to retest parts of systems that were not touched by system changes and confirm the new system parts integrate well with the old ones. This could for example happen in regression or "hardening" sprints.

In my view, this "after-testing" is a key area where it can pay off most to have, in advance, well developed test modules and fully automated actions resulting in valuable time savings, particularly if a release date is getting close. The test development and automation planning should address this use in final testing as a main objective, and identify and plan test module development accordingly.

Agile Test Automation

The term often used for test automation in agile projects that best describes what is needed is "just in time automation." When ABT is applied, the term changes to "just in time test development." Independent to that, a high level of automation can play an invaluable contribution in improving the productivity and speed in sprints.

To obtain the automation quickly and timely, a number of rules should be applied:

1. Build the base early
2. Make automation resilient
3. Address testability of the system under test
4. Test the automation

1. Build the base early

A successful automation architecture should start with creating a solid base on which further action can be developed. This includes items like the ability to perform all necessary operations on all UI interface classes, access to API's, ability to query databases, compiling and parsing messages in a message protocol, etc.

Although much technical functionality is available in LogiGear's TestArchitect tool, most of our projects will start with R&D efforts to address customer specific technical challenges, e.g. emulating devices in a point of sale system, working with moving 3D graphics for oil exploration, testing mobile devices, accessing embedded software in diagnostic equipment, etc.

This technical base is something to administer as soon as possible and as comprehensively as possible. Identify all technical challenges and resolve them. This typically results in the implementations for low level actions, that then in turn can be used for higher level actions, for example in development sprints. Addressing the technical base early also limits risks.

2. Make automation resilient

The essence of agile projects is that many details of the system under test only become clear when they are being implemented, as part of iterations like the sprints in Scrum. This holds in

particular for areas that automation tends to rely heavily on, like the UI. Those details can change quite easily as long as the creative process moves along. The automation should in such cases not be the bottleneck. Flexibility is essential.

The action model by nature can give such flexibility as it allows details to be hidden in individual actions, which can then be quickly adjusted if necessary. However, there are some additional items to take care of as well. The most common in our projects has turned out to be "timing." Often automation has to wait for a system under test to respond to an operation and get ready for the next one.

What we found is that the automation engineer should make sure to use "active timing" as much as possible. In active timing you try to find a criterion in the system under test to wait for, and wait for that up to a preset, generous, maximum. If the criterion is met, the automation should move on without further delay. Paying attention to these and similar measures will make the automation solid and flexible.

3. Address testability of the system under test

When preparing automation, system developers should identify items that the system under test should provide to facilitate easy access by automation. When the items have been identified early on and are formulated as requirements, the teams can easily incorporate it in the sprints.

A good example is the provision of values for certain identifying properties that are available in various platforms for screen controls or HTML elements, properties that are not visible to a user, but can be seen by automation tools. Providing such values will allow automation to address the controls or elements easily, and in a way that is usually not sensitive to changes in the design.

In fact if such values are defined early on in a project, a tool like TestArchitect allows for the creation of "interface definitions" to take advantage of them before the system under test is even built.

Examples of such useful properties are the "id" attribute in HTML elements, the "name" in Java/Swing, and the "accessibility name" in .Net and WPF. All of these do not influence the user experience, and can be seen by the tools. Using them also solves issues of localization: an *OK button* can be found even if its caption is in another language.

4. Test the automation

Automation should be tested. In ABT this means actions and interface definitions must be tested. They are like a product that automation engineers provide to testers, a product in which high quality is required. We require in each testing project to have at least one folder (in the TestArchitect test tree) with test modules that test the actions and interface definitions, not necessarily the system under test.

Just like the test development, the automation activities must be well planned and organized, and a number of experienced people to be involved. If that is the case the combination of careful test development planning and automation planning should be able to meet the demands of agile projects quite easily. ■

RELATED ARTICLE

Automation Testing Tools In Full View

Bogdan Bereza-Jarocinski takes a close look at test tool taxonomies with his in-depth analysis of the diverse options available to testers.

A trainer, quality specialist and owner of VictO (2004), Bereza-Jarocinski co-founded Swedish SAST (1995), Polish SJSI (2003), and Polish SPIN (2006). He is also founder and secretary general of ADPQB (2008). For more information on his academy, visit www.victo.eu .

In order to make the right choices among tools, you must be able to classify them. Otherwise, any choice would be at best haphazard. Without functioning classification, you would not be able to understand new tools fast, nor come up with ideas of using, or creating new tools. Taxonomy is a systematic description of tool features and therefore requires proper understanding of those features. Chaotic taxonomy means you do not really grasp the technology, or usage of such tools.

Existing Taxonomies

Software testing lacks standards, and software test automation lacks them almost completely.

- The section on testing tools in software testing chapter of Wikipedia[1] is very confusing–to say the least.
- ISO/IEC 29119 software testing standard[2] is under development and far from complete.
- Software process standards such as TMMi[3] or TPI[4] state their tool taxonomy only indirectly–by stating vaguely what types of test tools are required for various maturity levels.
- Maturity Model for Automated Software Testing (MMAST)[5] sounds promising, but is far from satisfactory, and almost totally unknown in software industry.

Last but not least, the most world-known de facto standard is ISTQB: International Software Testing Qualifications Board.[6] Its syllabi do offer a relatively comprehensive test tools classification, but it is long from sufficient to be useful in practice.

The categories used there: test management, test execution, debugging and troubleshooting, fault seeding, simulation, static and dynamic analysis, keyword-driven test, performance testing and Web tools[7] is more a mishmash of names belonging to different worlds (e.g. is keyword-driven tool something different than test execution tool?) rather than a useful taxonomy.

Tool Taxonomy for Agile Testing

Of course, a good and useful test tool taxonomy is the same for agile and for traditional development methods, but in agile methods you need it more, because product and project requirements change faster.

You cannot know all tools, but if you use a good taxonomy, than you have got a framework that enables you to locate and understand tools fast. Even a tool with a totally confusing name (that means, any tool—for some inexplicable reason both freeware and commercial test tools have names designed to impress and confuse, but never to help and guide the user), you are still able to understand roughly what it is in less than 30 seconds, provided you have the framework.

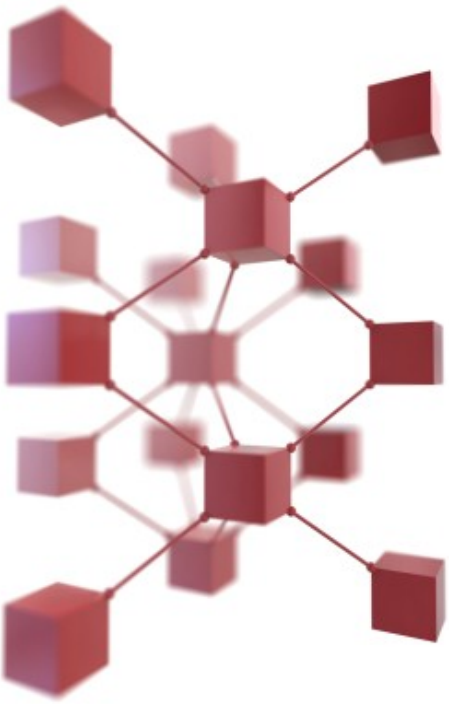
Using the framework presented below, you are no longer one-tool expert unable to see the context, nor you need be surprised by every new tool you encounter! But of course taxonomy does not replace detailed knowledge of given tools.

Outside the Framework

Some issues that are vital for understanding test tools do not fit nicely into tool taxonomy framework. We describe them shortly in this section.

- What test activities can be automated? The activities can range from intellectual to creative, and administrative to repeatable—you need to know which are best candidates for automation.
- Tools for testing products, projects and processes. Even if you seldom talk about project or process testing" (names like "audit" or "status monitoring" are used instead), such activities exist and are very important. In this article, we focus on product testing tools. But where does testing tools belong?
- Using tools is not yet automation. Tools must be used when human senses are not enough. Our eyes cannot see electrical or radio signals, and our hands cannot produce them. However, if you use a manually operated signal generator and a line listener, it is not yet test automation! This should be kept in mind.
- General types of test tools. Tools can be classified in a number of ways, but not all of them are useful in practice. However, you must be aware of these less important classifications, too, and not become confused.

  - Dedicated test tools and generic tools used for testing. The border-line is often unclear, but you must not mix up things. For example, a word processor used for writing test case specifications, or a spreadsheet used for managing bug reports are very useful for automating some aspects of testing, but no one would call them "test tools."
  - Test tools can be software applications (most common usage), mechanical tools, electric or electronic measurement instruments and generators (this includes tools more commonly used for HW testing such as oscilloscopes or logic analyzers), cameras or other tracking instruments (used often for usability testing), chemicals sensors, and many more.
  - Open source, freeware and commercial tools. Licensing schemes for commercial tools can be very different. Tool architecture and usage mode become more varied, too, as more and more cloud (software as a service, SAAS) tools appear.



  - The distinction between emulators, simulators, test environment (even for manual testing), and debuggers, is not always easy to make, nor clear-cut.

  - Tools with different degrees of intrusiveness from coverage measurement tools that instrument (change) the source code to electrical measurement instruments, whose only intrusiveness is probe effect in the form of resistance or inductance
  - The level of tool integration. Increasingly, tools are not used alone but with many other tools as part of an integrated environment for automated testing. This creates many new issues, which are not covered in this article.

It must be kept in mind that real tools often span across many categories of these classifications. Some of the categories can be argued do not belong to testing at all, for example debugging or configuration management.

Action-based Framework

The most obvious and useful classification is dividing tools by what they do. It is comfortable to identify four categories here: test management, test execution, testware preparation and special test tools.

Test Management Tools

Test project management tools

- Tools for test reporting

- Change and configuration management tools
- Incident management tools
- Tools for build and integration
- Requirement tracking tools

Test Execution Tools

- Functionality 1: the ability to start dynamic test (send input data, activate signal, invoke a procedure, press mechanical sensor); tool's "hands."
- Functionality 2: the ability to record actual test result (output data, activated signal, radio signal); tools "eyes."
- Functionality 3: the ability to compare actual and expected outcome; tool's "brains."

Typical test execution tools have all three functionalities, but there may well exist tools with only two of them or even one. Tools for dynamic analysis do not need any "hands," and not tools for static testing and static analysis, either.

Testware Preparation Tools

Test data preparation tools: such tools can prepare input data, or pairs input— expected output data for dynamic testing, or can be used for populating databases, and for many other purposes.

- Test document preparation tools: there exist tools for producing test specifications from requirements or design (model-based testing). Then there is the gray zone of test reporting and incident management tools (that belong as well to the test management tools family), or tools for creating test logs (part of test execution tools).
- Test script preparation tools:
  - Tools for script programming (with additional option for data-driven testing).
  - Capture tools (typically, as part of capture-replay tools).
  - Creating test scripts from source code (most commonly for unit testing); special cases here are automatic creating of test stubs and test drivers, and creating tests for markup languages (mostly HTML).
  - Keyword-driven script generation: creating scripts from test specifications written in formal language.
- Special tools: tools for test coverage measurement and for fault injection.

Goal-based Tool Classification

What are the test goals?

Security Testing

To some extent, any test tool can be used for security testing, but there are some specialized tools as well. Some are used for penetration testing–dynamic design and execution of tests addressing some known security problems. But even static code analysis tools often contain a large number of rules for security issues.

Safety Testing

There are few tools used specially for testing system safety. Some of them are hardware tools.

Performance Testing

Performance (load, stress) testing tools are test execution tools with special capabilities for creating big load and measuring performance parameters exactly. They come in many different versions and for various platforms, and their sales make today the biggest part of test tool industry.

Conformance Testing

Various types of dynamic and static test execution tools, that focus on whether system behavior conforms to legal, standard or industry rules.

Tools for Testing Other Quality Attributes

Since at least 50-100 quality attributes can be identified (depending on the classifications), many more tool types can identified. This is not very useful, however.

Platform-based Test Tool Classification

There are thousands of different platforms for software systems, depending on hardware, operating system, API, network type, programming languages and other aspects. Most questions asked about software tools concern this issue, which can hide other, core aspects of test tools. But they are very important in practice. Some sub-classifications here: Web tools (very broad category), language-specific tools, tools for different operating systems (including the growing OS flora for mobile devices—Android, Symbian, iOS…), tools for various HW platforms, tools for testing embedded systems.

Level-based Tool Classification

Bottom-up classification framework: hardware tools, debuggers, tools for static code analysis, unit testing tools and frameworks, tools for system and acceptance testing (including usability). ■

1 http://en.wikipedia.org/wiki/Software_testing#Testing_tools (all links checked on 12 March 2011)
2 http://softwaretestingstandard.org/
3 http://www.tmmifoundation.org/html/tmmiorg.html
4 See Test Process Improvement: A step-by-step guide to structured testing, Tim Koomen and Martin Pol
5 http://sqa.fyicenter.com/art/A_Maturity_Model_for_Automated_Software_Testing.html
6 http://istqb.org/display/ISTQB/Home
7 http://istqb.org/download/attachments/2326555/Advanced+Level+Syllabus+%282007%29.pdf

RELATED ARTICLE

Do Testers Have to Write Code?

With test automation now a more common practice, Elisabeth Hendrickson tackles the growing challenge of testers and their skill sets.

Do testers have to write code?

For years, whenever someone asked me if I thought testers had to know how to write code, I've responded: "Of course not."

The way I see it, test automation is inherently a programming activity. Anyone tasked with automating tests should know how to program. But not all testers are doing test automation.

Testers who specialize in exploratory testing bring a different and extremely valuable set of skills to the party. Good testers have critical thinking, analytical, and investigative skills. They understand risk and have a deep understanding where bugs tend to hide. They have excellent communication skills. Most good testers have some measure of technical skill such as system administration, databases, networks, etc. that lends itself to gray box testing. But some of the very best testers I've worked with could not have coded their way out of a For Loop.

So unless they're automating tests, I don't think that testers should be required to have programming skills.

Increasingly I've been hearing that Agile teams expect all the testers to know how to write code. That made me curious. Has the job market really shifted so much for testers with the rise of Agile? Do testers really have to know how to code in order to get ahead?

My assistant Melinda and I set out to find the answer to those questions. Because we are committed to releasing only accurate data, we ended up doing this study three times. The first time we did it, I lost confidence in how we were counting job ads, so we threw the data out entirely. The second time we did it, I published some early results showing that more than 75% of the ads requested programming skills. But then we found problems with our data, so I didn't publish the rest of the results and we started over. Third time's a charm, right?

So here, finally, are the results of our third attempt at quantifying the demand for programming skills in testers. This time I have confidence in our data.

We surveyed 187 job ads seeking Software Testers or QA from across the 29 states in the US posted between August 25 and October 16, 2010.

The vast majority of our data came from Craigslist (102 job ads) and LinkedIn (69 job ads); the rest came from a small handful of miscellaneous sites.

The jobs represent positions open at 166 distinct, identifiable companies. The greatest number of positions posted by any single company was two.

Although we tried to avoid a geographic bias, there is a bias in our data toward the West Coast. (We ended up with 84 job listings in California alone.) This might reflect where the jobs are, or it could be because we did this research in California so it affected our search results. I'm not sure.

In order to make sure that our data reflected real jobs with real employers we screened out any jobs advertised by agencies.

That might bias our sample toward companies that care enough to source their own candidates, but it prevents our data from being polluted by duplicate listings and fake job ads used to garner a pool of candidates.

Based on our sample, here's what we found: out of the 187 jobs we sampled, 112 jobs indicate that programming of some kind is required; an additional 39 jobs indicate that programming is a nice skill to have. That's just over 80% of test jobs requesting programming skill.

Just in case that sample was skewed by including test automation jobs, I removed the 23 jobs with titles like "Test Automation Engineer" or "Developer in Test." Of the remaining 164 jobs, 93 required programming and 37 said it's a nice to have. That's still 79% of QA/Test jobs requesting programming.

It's important to understand how we counted the job ads.

We counted any job ad as requiring programming skills if the ad required experience or knowledge of a specific programming language or stated that the job duties required using a programming language. Similarly, we counted a job ad as requesting programming skills if it indicated that knowledge of a specific language was a nice to have.

The job ads mentioned all sorts of things that different people might, or might not, count as a programming language. For our purposes, we counted SQL and shell/batch scripting as programming languages. A tiny number of job ads (6) indicated that they required programming without listing a specific language by listing broad experience requirements like "Application development in multiple coding languages." Those counted too.

The bottom line is that our numbers indicate approximately 80% of the job ads you'd find if searching for jobs in Software QA or Test are asking for programming skills.

Regardless of my personal beliefs, that data suggests that anyone who is serious about a career in testing would do well to pick up at least one programming language.

So which programming languages should you pick up? Here were the top 10 mentioned programming languages (including both required and nice-to-haves):

- SQL or relational database skills (84)
- Java, including J2EE and EJBs (52)
- Perl (44)
- Python (39)
- C/C++ (30)
- Shell Scripting (27) *note: an additional 4 mentioned batch files*.
- JavaScript (24)
- C# (23)
- .NET including VB.NET and ASP.NET but not C# (19)
- Ruby (9)

This data makes it pretty clear to me that at a minimum, professional testers need to know SQL. I will admit that I was a little sad to see that only 9 of the job ads mentioned Ruby. Oh well.

In addition, there were three categories of technical skills that aren't really programming languages but that came up so often

that they're worth calling out:

- 31 ads mentioned XML
- 28 ads mentioned general Web Development skills including HTTP/HTTPS, HTML, CSS, and XPATH
- 17 ads mentioned Web Services or referenced SOAP and XSL/XSLT

We considered test automation technologies separately from programming languages. Out of our sample, 27 job ads said that they require knowledge of test automation tools and an additional 50 ads said that test automation tool knowledge is a nice to have. (As a side note, I find it fascinating that 80% of the ads requested programming skills, but only about half that number mentioned test automation. I'm not sure if there's anything significant there, but I find it fascinating nonetheless.)

The top test automation technologies were:

- Selenium, including SeleniumRC (31)
- QTP (19)
- XUnit frameworks such as JUnit, NUnit, TestNG, etc. (14)
- LoadRunner (11)
- JMeter (7)
- Winrunner (7)
- SilkTest (6)
- SilkPerformer (4)
- Visual Studio/TFS (4)
- Watir or Watin (4)
- Eggplant (2)
- Fitnesse (2)

Two things stood out to me about that tools list.

First, the number one requested tool is open source. Overall, of the number of test automation tool mentions, more than half are for free or open source tools. I've been saying for a while that the commercial test automation tool vendors ought to be nervous. I believe that this data backs me up. The revolution I predicted in 2006 is well under way and Selenium has emerged a winner.

Second, I was surprised at the number of ads mentioning WinRunner: it's an end-of-lifed product. My personal opinion (not supported by research) is that this is probably because companies that had made a heavy investment in WinRunner just were not in a position to tear out all their automated tests simply because HP/Mercury decided not to support their tool of choice. Editorializing for a moment: I think that shows yet another problem with closed source commercial products. Selenium can't ever be end-of-lifed: as long as there is a single user out there, that user will have access to the source and be able to make whatever changes they need.

But I digress.

As long as we were looking at job ads, Melinda and I decided to look into the pay rates that these jobs offered. Only 15 of the ads mentioned pay, and the pay levels were all over the map.

Four of the jobs had pay ranges in the $10-$14/hr range. All four of those positions were part time or temporary contracts. None of the ads required any particular technical skills. They're entry-level button-pushing positions.

The remaining 11 positions ranged from $40K/year at the low end to $130K/year at the high end. There just are not enough data points to draw any real conclusions related to salary other than what you might expect: jobs in major technology centers (e.g. Massachusetts and California) tend to pay more. If you want more information about salaries and positions, I highly recommend spelunking through the salary data available from the Bureau of Labor Statistics.

And finally I was wondering how many of the positions referred to Agile. The answer was 55 of the job ads.

Even more interesting, of those 55 ads, 49 requested programming skills. So while 80% of all ads requested programming skills, almost 90% of the ads that explicitly referenced Agile did. I don't think there's enough data available to draw any firm conclusions about whether the rise of Agile means that more and more testers are expected to know how to write code. But I certainly think it's interesting.

So, that concludes our fun little romp through 187 job listings. I realize that you might have more questions than I can answer. If you want to analyze the data for yourself, you can find the raw data here.■

Elisabeth Hendrickson founded her company as Quality Tree Consulting in 1997 to provide training and consulting in software quality and testing. She incorporated the company as Quality Tree Software, Inc. in 1998. In 2003, Elisabeth became involved with the Agile community. In 2005 she became a Certified Scrum Master and in 2006 she joined the board of directors for the Agile Alliance. You can follow her blog at http://testobsessed.com/ or visit her company site Quality Tree Consulting at http://www.qualitytree.com/. To read the original post, visit http://testobsessed.com/?s=testers+have+to+write+code%3F

SPOTLIGHT INTERVIEW

Jonathan Rasmusson

Author of *The Agile Warrior*, Jonathan Rasmusson answers questions on agile testing from a developer's point of view

As an experienced entrepreneur and former agile coach for ThoughtWorks, Rasmusson has consulted internationally, helping others find better ways to work and play together. When not coaching his sons' hockey teams or cycling to work in the throes of a Canadian winter, Jonathan can be found sharing his experiences with agile delivery methods at his blog, The Agile Warrior*. Developer by trade, Jonathan Rasmusson engages with LogiGear's inquisitive staff to give us insight on how he addresses various agile test automation challenges. An excerpt from chapter seven of his book, titled "Planning Agile Projects: Estimation– The Fine Art of Guessing" will be included in next month's issue focused on Test Process Improvement. More excerpts can be found at The Pragmatic Bookshelf.*

1. How does test case automation fit in among different sprints? How can you automate testing on new functionality during a sprint? How do you make decisions during a spring to automate a test or not?

Generally when creating software in agile we like to do our testing in the same iteration (or sprint) that the story is developed in--this ensures that what we've produced works. It's more efficient than abandoning the story and picking it up again in the next iteration to complete the testing. So generally, it fits in towards the end of the iteration.

In having said that, getting all your automated testing done in a single iteration isn't always possible. For example, at the start of a new project we may not have a mature automated testing strategy in place. It's something we'll probably discover and figure out as we go. But at a minimum, the unit tests should be there along with whatever else we can get going early in the project.

Deciding which tests to automate (and which not to) is a decision every team faces on their project. Unit tests are cheap and easy - every project should be writing a ton of these. Integration tests are a bit more complex and involve making trade-offs between costs, scalability and fragility. For example a nice end-to-end integration or smoke screen test can be extremely valuable for a team that just wants to know whether the latest push of their software into a development environment is working or not.

Automated GUI tests on the other hand can be extremely fragile and problematic. It's not that you should never do them, you just have to be careful as you don't want to waste a lot of teams on fixing a changing UI test because the GUI hasn't settled down.

2. What type of test automation should we do in agile projects?

At a minimum, teams should be doing a lot of unit testing. These tests developers write to ensure that their software (at the method and class level) works as expected. Every team should be writing many of these.

After that, it really depends on the nature of your project and your team. If your system is primarily back-office, integration tests (tests verifying all your various subsystems) that are hooked up correctly can be extremely valuable. If you have a

GUI / web front end, you may want some high-level smoke screen tests using a tool like Selenium or something.

3. What are all the challenges for a test automation team in agile?

There are a couple of challenges with test automation on any team. One is having the courage and discipline to do it. Automated testing is hard work. It means not moving forward and delivering new functionality. It's hard for some people to stop, make the investment, and automate.

The second challenge of course is knowing what to automate. Teams need to figure and discover on each project which tests are going to pay off and which are going to give a lot of bang for your buck. We know we can't automate everything (that would be too expensive) but we also know that not automating is going to kill us. So we just need to be flexible, try lots of different tests, keep what works, and drop anything that gets in the way.

4. About test automation script maintenance, should it be considered as a separate user story/ sprint?

No. While it's OK for teams to reserve 10-20% of an iteration for general bug fixing, refactoring, or test script maintenance, I wouldn't recommend turning it into a story. Why? Because it's not something our customers would find very interesting. As much as we can, I prefer reserving stories for pieces of business functionality our customers would get excited about. Testing, refactoring, and general maintenance are generally things they don't care about.

5. What is the best time to document testing in agile projects, at the beginning from just the user story or at the end after the function is complete and fully understood?

While I like the idea of automating tests at the start of the sprint, I have never been able to pull it off in practice. I think you could do it if you had a really good idea of what was required and a team with the people, skillset, and desire to do it. I just haven't seen it yet. That would have to be a pretty mature and skilled team.

Also, quite often on new stories we don't know what the end result is going to look like. It's often not until the story is nearly complete, and we've gone through many iterations with the customer that we (including the customer) truly understand what is required—that makes automating ahead of time hard.

6. If other groups (marketing, development, project management) have cut back on documentation and measurement to become "leaner," should testers still be required to write test cases, show traceability to user stories, write a test plan and produce a dashboard of metrics on test cases and bug counts?

I don't see the connection. If one group in the organization is doing something different to become more lean or efficient—good for them. That's to be applauded. Conversely if other groups aren't being efficient or lean, that should stop us from doing whatever it is we feel we need to do to serve our customers better.

So if writing test cases, tracking traceability, and extensive test plans aren't adding any value, they should be dropped regardless of what others in the organization are doing.

7. If my team does not do a hardening sprint, when do we run a full regression test suite?

That's something each team needs to decide for themselves. While I am generally not a fan of hardening sprints, they are sometimes required. The team just takes on a lot of technical debt, or that they start slowing down because they haven't invested enough in their build and automation tools, so taking a sprint to firm these things up can help.

One obvious time to do a full on regression test is right before any major product release, e.g. a User Acceptance Testing (UAT). That's something you'll obviously want to do before going live.

Having said that, I believe the goal of every agile project should be to make UAT redundant or a non-event. In other words, the team does such a good job with it's automated and in house iteration testing, that when UAT comes around, no bugs are found; they're not there. The team has been building and releasing software of such a high quality that it is no longer required.

I am not saying drop UAT. You've got to earn the right to do that. But every day more and more teams are making UAT a non essential activity instead of spending that time and money on more valuable things (improving and adding more features to the existing software).

8. I am on an agile team that does offshore testing. Often, I do not attend the daily stand-up due to time zones. Can agile be successful this way? What are some suggestions to help me be successful in my testing?

I am no expert in offshore testing. Not really my specialty. But I do find it interesting, that whenever projects get into trouble, they bring everyone together and put them in the same room. This is often the single greatest thing you can do to improve the productivity of your team. ■

2010 GLOBAL TESTING SURVEY RESULTS

Automation Testing

Data was compiled and analyzed by Michael Hackett, LogiGear Senior Vice President. This is the first analysis of the 2010 Global Testing Survey. More survey results will be included in subsequent magazine issues. To read the overview, visit http://www.logigear.com/survey-response-overview.html.

The target audience of the survey were black box testers. Please note that to these respondents, test automation is mainly about UI level automation, not unit, performance or load testing.

These automation survey results contain two mutually exclusive sections. There were sets of questions for teams that currently *automate* tests and another set for teams that currently *do not automate* any tests.

I. Overview

Before delving into the respondent's frame of mind with answers to questions from *Test Automation*, I will highlight some results from *The Politics of Testing*, *Training*, *Strategy* and *Overview* sections that will set the stage for a better understanding of the issues faced by these respondents in test automation.

PT1 (Politics of Testing)- What phase or aspect of testing do you feel the management at your company does not understand? (You can select multiple answers.)

Projects are most often behind schedule because of shifting requirements not test delays.	43%
How to measure testing	41%
How to adequately schedule testing	41%
Test automation is not easy	40%
Testing requires skill	35%
The inability of complete testing	32%
Choosing good coverage metrics	23%
None, my management team fully understands testing.	17%

Result analysis: The 3rd highest response, virtually tied as the area of testing that management does not understand test automation is not easy!

PT2- What challenges does your product team have regarding quality? (You can select multiple answers.)

Insufficient schedule time	63%
Lack of upstream quality assurance (requirements analysis and review, code inspection and review, unit testing, code-level coverage analysis)	47%
Feature creep	39%
Lack of effective or successful automation	36%
Poor project planning and management	33%
Project politics	31%
Poor project communication	27%
Inadequate test support tools	25%
No effective development process (SDLC) with enforced milestones (entrance/exit criteria)	25%
Missing team skills	23%
Low morale	16%
Poor development platform	8%

Result analysis: By far, the #1 answer here is insufficient schedule time. The #4 answer is a lack of successful automation. It is too easy to say more investment in test automation will solve all your team's problems—but it will definitely help! More and more effective test automation always helps projects.

It is not the answer to *all* problems, as clearly emphasized in the second highest choice, that a lack of upstream quality practices cannot be solved by downstream test automation! But better test automation will go far in helping the manual test effort and by doing so, at least relieve some tester stress.

T1 (Training)- What skills are missing in the group? (You may select multiple answers.)

Automation skill	52%
Coding/tool building	42%
Technical understanding of the code, system, platform, environment	35%
Subject matter/domain knowledge	33%
QA/testing skill	32%
Test tool (ALM, test case manager, bug tracking, source control, continuous integration tool)	22%

Results analysis: Interesting but never surprising, the highest chosen answer by teams in regards to what they lack—more than half the respondents—is test automation skills! It is obvious and clear that acquiring more test automation skills is the single most important job security point.

S1 (Strategy)- How do you do regression testing?

Both	47%
Manual	34%
Automated	15%
We do not do regression testing	4%

Results analysis: A very big surprise to me—the lack of automated regression! Wow. That is one of the biggest and most surprising results of the entire survey! Why do 1/3 of teams *still* do all manual regression? Bad idea, bad business objective.

O1 (Overview)- Do you currently automate testing?

Yes		63%
No, not currently		37%
If no, has your team ever automated testing?	Yes	90%
	No	10%

Results analysis: With over 1/3 of respondents currently not automating tests, these results, however, are contrary to popular belief and any sort of *best practice.* What I see out in the business world are many teams that think everyone automates and they themselves automate enough.

I also see many teams where all testing is manual and see automation as *not* common, too difficult, and not something *testers* do. This number is alarmingly high. Any team not automating has to seriously look at the service they are providing their organization as well as the management support they are receiving from that organization!

II. For teams that currently *automate testing*

A1 (Automate)- Have you been trained in test automation?

Yes	70%
No	30%

Results analysis: For teams that currently automate and still have 30% of the team untrained in test automation is deeply problematic. When this is the case, I often see the problem as too much technical information centralized into too few people. This is problematic for the career growth of staff, a business analyst or subject matter expert team as it demonstrates management does not invest in its staff. If your team automates and you have been left behind, it is a good idea to get a book, take a class, educate yourself and insinuate yourself into test automation for your own career planning!

A2- Estimate what percentage of the test effort is automated?

Less than 25%	36%
50 – 74%	32%
Over 75%	14%
25 to 49 %	13%
Very little.	5%
All our testing is automated	0%

Results analysis: Still amazing to see how little test groups automate. Over 40% of teams automate less than 25% of their tests! With 46% of teams automating over 50% of their test effort reveals very significant strides can be made in reducing the dependence on manual testing.

A3- How would you describe your automated test effort?

Is it a very effective part of your test effort.	33.30%
It is key to meeting schedule dates. Our test schedule would be much longer without automation.	19%
It is key to improving product quality.	14.30%
It is key to product stability.	9.50%
It is effective for just what we have automated and does not significantly affect the rest of the test effort.	9.50%
It frees me up from repetitive tasks to do more interesting, exploratory testing.	9.50%
It is somewhat useful, but has not lived up to expectation or cost.	4.80%
It is the most important part of our test effort.	0%
It is a waste of time and does not tell us much.	0%

Results analysis: What surprises me most about this set of answers is that no one thought the automated tests were the most important part of their test effort. What does this say? The respondents take their automation for granted? It isn't trusted or it isn't good? Or quite possibly, it is important but having a human manually interact with the system is the most important part of the test effort! Exactly 1/3 said it was a very effective part of the effort. Almost 20% said it is key to meeting the schedule. The fact that these numbers are not higher shows how test automation has not yet achieved its full potential in helping teams.

A4- How do you measure test automation effectiveness?

Time saved from manual execution.	81%
Percentage lines of code covered by automated tests.	14.30%
Number of bugs found.	4.80%
Product stability measured by number of support/help desk calls from users or hot fix/patch fixes.	0%
We do not measure test effectiveness for automation	0%

Results analysis: This is an overwhelming measurement of test automation by time saved from manual testing. An important observation to note is that we don't measure test automation by bugs found- it's important for teams to understand this and clearly they do.

A5- What strategy do you use to design your automated tests?

Direct scripting of manual test cases	33.30%
Data driven	19%
Action based/keyword	19%
Record and Playback	14.30%
No specific method	14.30%

Results analysis: These results are interesting to see the level of sophistication of various teams' efforts.

A6- What is the main benefit of test automation?

More confidence in the released product	42.10%
Faster releases/meeting the schedule	36.80%
Higher product quality	10.50%
Waste of time	5.30%
None/no benefit	5.30%
Finding more bugs	0%
More interesting job/skill than manual testing	0%
Finding less bugs	0%
Less focus on bug finding	0%
Slower releases	0%

Additional useful comment from a respondent: "Some of the best uses of automation are to exercise the application, perform redundant or error-prone tasks, configure or set up test environments, and to execute regression tests."

Results analysis: The results are encouraging for automation—there is a consensus that testers believe test automation provides more confidence in the quality of the product and increases the ability to meet schedules.

III. For teams that *do not automate* testing

A7- Have you tried to automate tests for your product?

Yes	56%
No	44%

Results analysis: A surprising number of teams have never tried automating! I think this is another dark secret of software testing. Drawing from my own speculation, it's my opinion that many companies either have never invested in test automation, do not realize its benefits, afraid to try something new, or realize they need a significant investment to make it work and are not willing to further fund testing. It could also be that teams may have tried automating and given up, were not supported by the development organization, or test tool funding was cut. These are situations that need to be addressed for test teams to provide long term benefits to the organization.

A8- What would prevent you from automating tests now? (You may select multiple answers.)

Investment in automation program of training, staff, tool, and maintenance cost is too high.	43.80%
Tool cost too high.	37.50%
Management does not understand what it takes to successfully automate.	37.50%
Not enough technical skill to build successful automation.	37.50%
Code or UI is too dynamic to automate.	37.50%
Test case maintenance cost too high.	25%
Not enough knowledge about how to make automation effective.	25%
It will not help product ship with better quality or in shorter schedule.	25%
Bad project management will not be helped by automating tests.	12.50%

Results analysis: The great variety of reasons why teams do not automate is clear: cost, management misunderstanding of automation and lack of knowledge are the great downfall of test automation.

A9- Are you, or is your current team technically skilled to the point where you can build and maintain test automation? (Remember, this response is only from teams *currently not automating.)*

No	56%
Yes	44%

Results analysis: If you are a software tester without much knowledge about automation, it would be best for your own career security to dive into test automation, read as much as you can, learn about the best methods, and see what various tools can actually do. Take responsibility to learn about the future of your career.

A10- Would product quality improve if you automated tests?

Yes	69%
No	31%

Results analysis: It is problematic that 31% of respondents do not see product quality improving with automation. Some teams may think the quality of their product is high and they do not need to automate. For those teams not so optimistic, there are a few possible ideas behind this: not understanding what information automated tests do and do not give you, not understanding tasks that can be automated to free up time to do more, deeper manual testing, but also, some teams may be resigned to low quality products, regardless of how their testing gets done.

IV. Participant Comments

The following statements are comments from respondents and their experience with test automation:

1."Make sure the test cases that are designated for automation are of the right quality."

2."I have oh so many. I worked for a test tools company for over 8 years. How much time do you have? ;-) Common problems were inflated expectations of automation, lack of skills to implement effectively, lack of cooperation with the dev teams in enabling testability features, etc.

But the worst stories I have are related to an over-reliance on automation where tools replaced people for testing and products were shipped with confidence that failed in the field horribly upon first use. These scenarios were VERY embarrassing and cause me to often throw the caution flag when I see a team driving toward 'automating everything.'"

3."Automation frees up resources to do more stuff. Instead of spending 5 days running a manual regression the automated regression runs in 1/2 day and the team can spend a day doing data validation and exploratory testing."

4."Test automation requires collaboration between developers, automation engineer, and functional test engineer. The more transparent the automation effort is, the more useful it will be."

5."Identify the appropriate tools for your project. Use it everywhere possible, when in the testing"

6. "I'm not sure if this is experienced by testers worldwide, but I had several encounters of IT Project Managers having the misconception of test automation. They have the expectation that every functionality in the application should be fully automated.

As a test automation engineer, I've [learned] that in an application, not all functionality could be automated due to limitations on the automated tool. Or the application function is too complex to automate, producing an ineffective test automation effort. My strategy to overcome this is to advise the manager to identify the critical functionalities that can be effectively automated, thus reducing manual testing effort significantly and reaping the most out of the automated tool." ■

Next month's survey analysis is on Test Strategy and SDLC.

VIET NAM SCOPE

McAfee Visits Ho Chi Minh City

McAfee staff from India and Japan visited for a week and left reflecting upon Viet Nam's culture and its future.

Lolita Guevarra

McAfee representatives from India and Japan visited LogiGear Viet Nam from Feb 21 - 25. Their week-long stay entailed debriefing LogiGear staff on McAfee MX Logic and Mobile Security for the team to begin testing. Project manager Tien Nguyen and project lead Hung Le oversaw the sessions between LogiGear and McAfee.

As a good hostess, Nguyen made sure between meetings that Masaki Sumida of Japan and Gargeya Doddaiah of India maneuvered throughout Viet Nam's infamous traffic to expand their palette at local eateries and awaken their senses within the country's vibrant culture.

Left to right: Masaki Sumida, Hung Le and Gargeya Doddaiah

Although the representatives have long returned to their respective countries, Viet Nam's stamp has made a lasting impression for Masaki who enthusiastically shares his experience with *LogiGear Magazine*:

1. What is it that you do for McAfee?

[I work in] Deployment and QA for Consumer Products.

2. What was the main objective of your visit?

We are working with the team in Viet Nam on several mobile projects. [The] main purpose of the visit was to give the team product demos, discuss test plans and do team building.

3. What was your experience like working with LogiGear Viet Nam staff?

I'm glad to work with the team. They enjoy the challenge. Everything is new and exciting to me.

4. What are your thoughts on Viet Nam being the next big tech outsource country? It has been called the "next India," any thoughts on that label?

I feel that young people in Viet Nam are leading the team and country. Big energy in the country—this is the biggest advantage and [the youth] respond flexibly to everything.

5.What kind of impression did Viet Nam leave on you? What stood out the most during your visit in Viet Nam?

Bikes everywhere. I saw it [in] pictures and books, but [it] was a surprise indeed. [I am] definitely coming back to Viet Nam. Not only [for] business, but also [as a] private trip.■

NEXT MONTH'S ISSUE

Test Process Improvement

Feature Article
Himanshu Jain is a test engineer at Hewitt Associates in India. He maintains a blog highlighting the latest trends in manual and automated testing at Fun With Technology.

Book Review
Bryan Pendleton is a software engineer living in northern California. He works for Perforce Software as a server developer, and is also a committer on the Derby project at the Apache Software Foundation. Read his blog Journal of a Programmer.

Blogger of the Month
Rob Lambert blogs regularly at The Social Tester, is the Creative Director at The Software Testing Club and is the Test Manager at NewVoiceMedia.

LogiGear®
Software Testing

LOGIGEAR MAGAZINE
MARCH/APRIL 2011 | VOL V | ISSUE 3

United States
2015 Pioneer Ct., Suite B
San Mateo, CA 94403
Tel +1 650 572 1400
Fax +1 650 572 2822

Viet Nam, Da Nang
7th floor, Dana Book Building
76-78 Bach Dang
Hai Chau District
Tel: +84 511 3655 333
Fax: +84 511 3655 336

Viet Nam, Ho Chi Minh City
1A Phan Xich Long, Ward 2
Phu Nhuan District
Tel +84 8 3995 4072
Fax +84 8 3995 4076