TestArchitect™ by LogiGear
COLLABORATIVE TEST AUTOMATION
Free 30-day trial
Learn More

DEDICATED TO SHOWCASING NEW TECHNOLOGY AND WORLD LEADERS IN SOFTWARE TESTING

LogiGear MAGAZINE

THE TEST PROCESS IMPROVEMENT ISSUE

Get High Performance Out of Your Testing Team!

By Michael Hackett
Senior VP
LogiGear

BOOK REVIEW

How We Test at Microsoft
Reviewer: Bryan Pendleton

RELATED ARTICLE

Agile Retrospectives: The Preventative Medicine
By 3months

SPOTLIGHT INTERVIEW

Mark Levison,
Agile for Pain Relief Group

EDITOR'S LETTER

EDITORIAL STAFF

Editor In Chief
Michael Hackett

Managing Editor
Lolita Guevarra

Contributing Editor
Thi Doan

Designer
Dang Truong

Webmaster
Bao Tran

Worldwide Offices

United States Headquarters
2015 Pioneer Ct., Suite B
San Mateo, CA 94403
Tel +01 650 572 1400
Fax +01 650 572 2822

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

www.logigear.com
www.logigearmagazine.com
www.logigear.com/blog

Copyright 2011
*Logi*Gear
All rights reserved.
Reproduction without permission is prohibited.

Submission guidelines are located at
http://www.logigear.com/logigear-magazine/submission-guidelines.html

On the whole, everyone wants to do a great job, have a better work environment, happy clients and customers, and to be employed by a company earning lots of money. All great goals! But this is not always the case. When it is not, you can suggest process improvements, better tool use, different estimating techniques, etc. Suggestions are generally evaluated based upon whether they are opinions, complaints, thoughtful, useful, possible or even mean-spirited.

A large part of my work over the past few years has been consulting on process improvement for entire software teams or specific test groups helping companies and teams achieve their goals. This all sounds great but I have to say—to achieve meaningful change is often painful! My process improvement work includes team skill assessments, team practice or tool use evaluations. What do I see? Lately, it is companies saying they are *Agile* when they are not; development teams not taking advantage of the recent huge growth in testing tools; and the traditional favorites: bad requirements dooming a project and unreasonable schedules.

We all want to improve but change is sometimes difficult. There are ways to do process improvement well and ways to set-off turf wars. This is why we included the theme of process improvement on the editorial calendar this year. We hope to provide you with insight and experience into how and what to do to make meaningful and beneficial changes leading to happier teams and customers along with less stress, lower costs and waste reduction.

Our test process improvement issue includes my piece centered on key tips on how to get high performance out of your test teams; two articles on Agile Retrospectives from New Zealand Agile consultant 3months and Mark Levison from England, who is also this month's Spotlight Interview; an additional related article focused on Retrospectives and Post-mortems and whether or not they are more similar than different; Blogger of the Month Rob Lambert links test process improvement to his encounter at an electronic store; Bryan Pendleton reviews *How We Test at Microsoft*; and the fourth part of the 2010 Global Testing Surveys series with analysis on Test Process and SDLC.

LogiGear magazine continues to be a resource for your software development projects and company needs—good luck!

Michael Hackett
Senior Vice President
Editor In Chief

IN THIS ISSUE

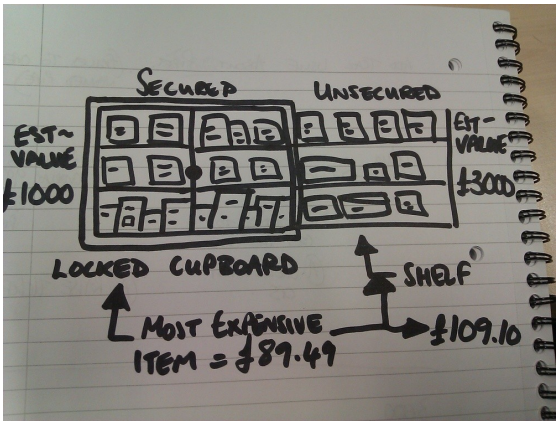
BLOGGER OF THE MONTH

Rob Lambert: The Social Tester

Creative Director at the Software Testing Club, Rob Lambert always has something to say about testing. Lambert regularly blogs at TheSocialTester where he engages his readers with test cases, perspectives and trends.

"Because It's Always Been Done This Way"

Study the following (badly drawn) image and see if there is anything obvious popping in to your head.



Now, you've probably got hundreds of questions going through your mind about testing, etc. But I want to draw your attention to the security and the value that this security is protecting. Let me backfill the story.

I went to a major electrical goods chain here in the UK. I wanted to buy a hard drive. I saw the one I wanted, behind the glass in the locked and secure glass cabinet. I asked the assistant if I could have one and it took her about 2 or 3 minutes to find the keys. It then took her about 2 minutes to open the cabinet, get the disk out and then lock it all back I then scanned just the two meter shelf next to the cabinet and it had about £3000 of Routers and Switches.

The glass cabinet had items that ranged between £30 and £89.49. Yet the shelf had items that ranged between £30 and £109.10.

It didn't really make sense to me. If I were the store manager, I would be looking to secure the most expensive and valuable stock. Wouldn't you? So I asked the lady why this was the case.

She said "Because it's always been done that way. I guess I'd never thought to question it".

I asked whether hard drives were the most stolen of items, fishing to see whether high targeted products were being secured over high value products. The lady didn't know. I tried to find some stats but failed. I suspect it's not got much to do with it.

The lady said that many displays, concepts and ideas came from head office and remained that way even if they were ineffective. It's because it's always been done like that... I guess.

And so I draw the comparison to testing, product design, feature sets and anything else we may ever get involved with in our daily work.

Why keep doing something when it is ineffective? Why spend months writing test documentation that no one reads? Why spend months writing test cases in minute detail to have to re-write them when you see the final product? Why always concentrate your security testing on the software when in reality the user is the biggest security gap? Why keep reporting that metric when no-one needs it?

Why not challenge the norm? The ineffective? The old rules and processes?

Why not suggest new and interesting ways of doing things? Why not improve or tweak processes for changing circumstances?

"Because it's always been done that way".

Do you have compelling reasons to leave something ineffective in place? Please let me know. ■

Email Rob Lambert at robk@thesocialtester.co.uk

FEATURE ARTICLE

Get High Performance Out of Your Testing Team

Six key phases are listed by Michael Hackett for test process improvements that require planning, execution and evaluation at all company levels.

Testing is often looked upon by many as an unmanageable, unpredictable, unorganized practice with little structure. It is common to hear questions or complaints from development including:

- What are test teams doing?
- Testing takes too long
- Testers have negative attitudes

Testers know that these complaints and questions are often unfair and untrue. Setting aside the development/testing debate, there can always be room for improvement. The first step in improving strategy and turning a test team into a higher performance test team is getting a grasp on where you are now. You want to address the following:

- What type of testing is effective?
- Are we testing the right things at the right time?
- Do we need a staffing upgrade?
- What training does our team need?

How does the product team value the test effort?
In this article, we provide a framework for assessing your team: planning your assessment, executing the assessment and judging your current performance, using the information, and charting an improvement plan towards higher performance.

The Test Process Assessment

The goal of doing a test process assessment is to get a clear picture of what is going on in testing: the positive aspects, the problems, and the possible paths to improvement. Fundamentally, a test assessment is a data gathering process. To make effective decisions we need data about the current test process. If done properly, the assessment will probably cross many organizational and management boundaries. It is important to note when embarking upon such an assessment that this effort is much larger than the test team alone. Issues will arise over who owns quality as well as what is the goal of testing? It is also important to note that a possible result of the assessment is that work may actually increase. Some issues that may arise are:

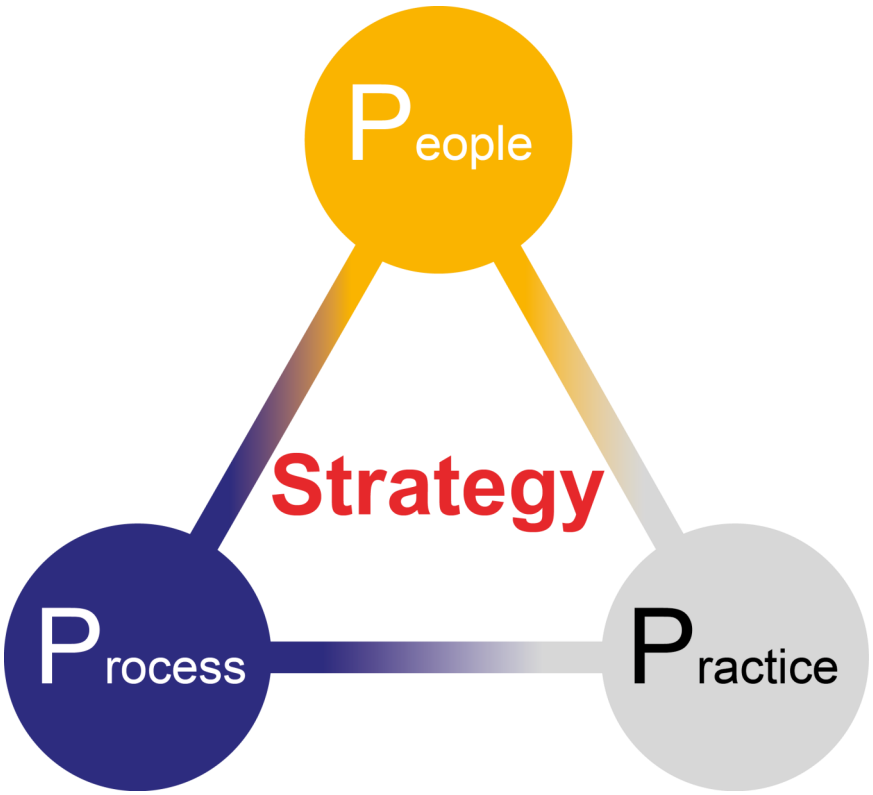
- More demands for documentation
- More metrics
- More responsibility for communication and visibility into testing.

For such an assessment process to succeed requires:

- Executive sponsorship
- A measurement program
- Tools to support change
- An acceptance of some level of risk
- Avoidance of blaming testing for project-wide failures
- Commitment about the goal of testing
- An understanding of testing or quality assurance across the product team
- Responsibility for quality

Components of a Test Strategy - SP3
A test strategy has three components that need to work together to produce an effective test effort. We developed a model called SP3, based on a framework developed by Mitchell Levy of the Value Framework Institute. The strategies (S) components consist of:

1. People (P1) - everyone on your team
2. Process (P2) - the software development and test process
3. Practice (P3) - the methods and tools your team employs to accomplish the testing task



Phase 1: Pre-Assessment Planning:
The goals for this phase are to set expectations, plan the project, set a timeline, and obtain executive sponsorship. The actions that occur in phase one include meeting with the management of various groups, laying out expectations for the results of the process, describing the plan, and establishing a timeline.

The intended result is to obtain agreement on expectations and buy-in on the assessment process and the follow-up commitment for improvement. The phase one deliverable is a schedule and a project plan. It is important at this stage to:

- Get executive buy-in
- Create a schedule and adhere to it
- Give a presentation of your plans discussing the objectives
- State goals or outline work as a commitment
- Make a scope document a pre-approval/budget deliverable

It is important to note up front that assessment is only the beginning of the process.

Phase 2: Information Gathering
The goal of phase two is to develop interview questions and surveys which become the backbone of your findings. Actions in phase two include gathering documentation, developing interview questions, and developing a test team survey.

The result of this phase is that you will be ready to begin your assessment of the materials.

Examples of the documentation to be collected include: SDLC, engineering requirements, and testing documents (e.g. test plan templates and examples, test case templates and examples, status reports, and test summary reports).

Interview questions need to cover a wide range of issues including (but not limited to): the development process, test process, requirements, change control, automation, tool use, developer unit testing, opinions about the test team from other groups, expectation of the test effort, political problems, communication issues, and more.

Phase 3: Assessment
The goal of phase three is to conduct the interviews and develop preliminary findings. Actions include gathering and reviewing documentation, conducting interviews, and distribution and collection of surveys. As a result of this phase there will be a significant amount of material and information for review.

Phase 4: Post-Assessment
Synthesize all information into a list of findings in phase four. Actions include reviewing, collating, analyzing, and making postulations. The result of this phase is that you will develop a list of findings from all of the gathered information, reviewed documentation, interviews, and the survey. Phase four deliverable is a package of collated survey answers, interview responses, a staff assessment, and a test group maturity ranking.

The findings can be categorized into:

- People
- Technical skills
- Interpersonal skills
- Process
- Documentation
- Test process
- SDLC
- Practice
- Strategy
- Automation
- Environment
- Tools

More subcategories may also be developed to suit your needs.

Phase 5: Presentation of Findings with Project Sponsor, Executive Sponsor and Team
Phase five presents preliminary findings to executives and the project sponsor, and to obtain agreement on the highest priority improvement areas.

It is important in this phase to be prepared for a very different interpretation of the findings than you perceived. The deliverable for phase five is an improvement roadmap.

Phase 6: Implementation of Roadmap
Phase six establishes goals with timelines and milestones and sub tasks to accomplish the tasks agreed upon for improvement. The action of phase six is to develop a schedule for implementation of the improvement plan.

It is helpful at this point to get some aspect of the project implemented immediately so people can see tangible results right away—even if they are the smallest or easiest improvement tasks. The deliverable for phase six is implementation of items in the roadmap according to the developed schedule.

Is it different for Agile?

Yes.

Process improvement is built into most agile processes. Scrum, for example, has sprint retrospectives. If you are not doing sprint retrospectives, you are not doing Scrum. This practice serves to foster very regular self-examination to quickly fail and immediately fix any efficiency. Remember the Scrum Master's primary objective is removing obstacles from people's work.

An Agile approach to product development relies on self-forming teams, always optimizing a dynamic process. With a regular and quick process improvement built into most agile practices, large scale assessment and improvement processes do happen, however less frequently than in traditional projects. The new name for a "process consultant" is a scrum coach.

Conclusion
A test strategy is a holistic plan that starts with a clear understanding of the core objective of testing, from which we derive a structure for testing by selecting from many testing styles and approaches available to help us meet our objectives.

Performing an assessment helps to provide "clear comprehension" and "understanding of the core objective of testing." Implementing the resulting roadmap for improvement can help to substantially improve the performance of your software testing organization and help to solidify your test strategy. ■

To contact Michael Hackett, email him at MichaelH@logigear.com. For more information, please visit www.logigear.com.

RELATED ARTICLES

3months, a consulting group, explains how the frequency of retrospectives encourage efficiency and flexibility in Agile and Scrum.

One of the features of using Agile methods is the opportunity for continuous improvement within a project. There are a number of improvement opportunities throughout a typical iteration or sprint—over the next few weeks I'm going to walk through a few, starting this week with the Retrospective. Retrospectives are one of the many tools in Scrum and other Agile methods that are absolute must-haves for continuous improvement.

Think about the last time you built a product (or had one built for you). If you weren't using an Agile method like Scrum, chances are that you pushed through the project in one big rush, waiting until the end to think of the things you would do differently next time. If you were lucky there may have been a chance for a review afterwards.

Enter the retrospective. Retrospectives take place frequently, at the end of every sprint (typically 2-4 weeks), and give the team a big opportunity to do things better sprint by sprint. A retrospective helps the team identify any changes or corrections needed during the project.

While I'm a fan of a good post-project review it is often equated to a post-mortem and, following this analogy through, retrospectives are closer to the idea of preventative medicine. Ultimately this isn't so different from any other lessons learned process but for me the frequency and drive for action that comes out of a retrospective make them highly effective. As an added plus, retrospectives are also fun to facilitate (if you're into that sort of thing) and generally make everyone feel more positive about the approaching sprint.

Running a retrospective is a pretty standard process but again is open for improvement and adaptation. My favored format is:

1. What just happened? (aka "set the stage") The team builds a brief time-line of what happened during the sprint.
2. How are we feeling? Each team member draws a mood line of how they were feeling throughout the timeline, talking through their personal peaks and troughs during the sprint (I'd love to know how this technique came about!)
3. Things to note! You can use whatever categories you need here, but generally Good Things, Bad Things, and Ideas work well. Some facilitators also add Bouquets—things that other team members have done during the sprint that are worthy of praise.
4. Actions: The best way to get a short-list of actionable items is to use dot-voting/multi-voting to highlight the top three or four things that the team wants to address. If these items are related to the product they're moved into the product backlog. If they are process related they generally become the project lead/Scrum Master's (or sometimes Product Owner's) responsibility.
5. Close the retrospective: Check how the retrospective went, and confirm the next steps.

For a bigger retrospective, particularly in between project phases, I think it's good to use a more elaborate Innovation Game to generate ideas for how things could be better next time. I like Remember the Future—it's a game that can be used for either product brainstorming or focused towards the effectiveness of the whole team.

Retrospectives are a good habit to get into. They can also be used in non-Agile delivery as well (i.e. in Waterfall, running a retrospective after the completion of every milestone).

More on retrospectives

There are some good books (Agile Retrospectives) and many articles around the web but ultimately you need to run one, get feedback from the team and start improving. Some useful starters:

- An action planning focused method
- Another on product level retrospectives
- A useful article on improving your retrospectives—you'll get the idea soon enough. ■

To read more please visit 3months.

AGILE RETROSPECTIVES

Mark Levison discusses advantageous attributes of the method.

Continuous Improvement and Short Feedback loops (think: Test Driven Development; Sprint Demo/Review; …) are at the core of any Agile process. Without a structured improvement process it can be difficult for teams to improve and without improvement we stagnate. For methods like Scrum, XP and et al., Retrospectives are that tool.

What is a Retrospective? It is a moment for the team to stop, breathe and take a break from the day to day grind. It's a chance to step back and reflect on the past iteration. To find things that worked well, things that need improvement and what the team has the energy to improve.

How do Retrospectives differ from Post-mortems (see CIO Update and PragmaticSW)?

- Post-mortems occur after the project is done (or even dead), when it's too late to improve that project.
- Post-mortems are long feedback loops, once per project might mean every 6-18 months.
- Post-mortems often generate nice reports that are placed on a shelf and ignored (also called write only documentation).
- Post-mortems sometimes turn into blame and shame events.

Well run retrospectives provide an opportunity for small improvements. The keys to a well run Retrospective:

Retrospective Prime Directive (Norman Kerth): "Regardless of what we discover, we understand and truly believe that everyone did the best job they could, given what they knew at the time, their skills and abilities, the resources available, and the situation at hand." *The key here is to remind participants at the start of every retrospective this is not to blame and shame. It's about understanding what happened in the course of the last iteration. The focus is on events and not the people.*

- A clear agenda – a simple one:
- What happened in the last iteration (including what SMART goals did we achieve?)
- What would like to be celebrated/remembered/…
- What areas need improvement?
- What improvements should we put our energy into for the next two weeks?
- Clear Ground Rules see: Meeting Ground Rules Updated
- An Open Mind from all team members with the focus on solutions and not just the problems.
- Appreciations – just take a few minutes to share something that you really appreciate that someone else on the team did. *Interesting twist I just noticed that Ellen Gottesdiener puts appreciations at the beginning of her Retrospective. That's very interesting, that will help put people in a positive frame of mind and make it easier to tackle the problems later. Elegant.*
- Once you decide what you have the energy to tackle, set SMART Goals (Specific, Measurable, Attainable, Realistic/ Relevant, Timely). See: SMART Goal Setting. In the context of an Agile/Scrum team I would always make timely less than two weeks, so that you check back in the next retrospective.
- A great facilitator who is able to stay out of the conversation and maintain the flow. Bring an outsider in occasionally, just to see a different approach.
- Mix-up your retrospective activities to maintain the energy and interest:
  - *Agile Retrospectives: Making Good Teams Great*, by Diana Larsen and Esther Derby
  - "Timelines and Mad/Sad/Glad": (lots of other good retrospective ideas, too)
  - SaMoLo (same of;More of; Less of)
  - Retrospective Wiki

Follow-up:

- Post your SMART goals on the team's Information Radiator and check up on them in the Daily Scrum.
- If you don't make the improvements that people choose then Retrospectives will quickly lose their value as people say: "Nothing ever happens from these".

When: At the end of every iteration or sprint. Allow one hour for every week of iteration. So a 1 week sprint would have 1 hr, 2 weeks a 2 hour retrospective. 3 weeks a 3 hour retrospective, 4 weeks – no one does those anymore right?

Who: The whole team – I like to see (or hear) the Product Owner and the Scrum Master. Some people will tell you that the PO isn't necessary. Fine, but if they're not there they can't help make things better. ■

RELATED ARTICLE

Same or different?

Michael Hackett provides seven distinctions between retrospectives and post-mortems.

Let's look at a few distinctions between the two process improvement practices that make all the difference in their usefulness for making projects and job situations better! An extreme way to look at the goals of these practices is: what makes your work easier (retrospective) versus what did someone else decide is best practice (post-mortem)?

First, look elsewhere for a "how-to." There are many published materials on how to do a sprint retrospective and a post-mortem. We are looking at the differences in the mindset. To look at how post-mortems and retrospectives are different, let's go back to the approach and foundation for these meetings. The following distinctions between the Agile and traditional software development mindset loom large in the mindset entering retrospectives and post-mortems.

1. **Chickens and Pigs vs. Top Down Management**
   The chickens and pigs story illustrates that in Agile, the working team (pigs), not management (chickens) dictates what the team does. Only the people who have skin-in-the-game decide what to do. Ideally, what this team wants to do and how it wants to do it falls on the team. In post-mortem, a "Top 10" style problem list is drawn up that later needs to be approved by other people—not the team members or those doing the work, then scheduled (or not) by other people, and budgeted by non-team members, who will decide for the team if they like the idea. This is always a tough part of the process—"selling" change to management, even though management will never work under the conditions you are suggesting to fix.

2. **Lean Manufacturing Principles**
   In most agile practices, there are few to no "process" documents. What documentation that does exist will be lean or light. In sprint retrospectives there is little documentation to deviate from or reference against. Post-mortem meetings can usually point to documents, approvals and sign-offs that are service level agreements (SLAs) teams agree to. If these agreements are not met, delayed, incur problems, then the documents will help "make your case."

   Lean principles make change more agile, dynamic, but have drawbacks of few reference points and often leave big gaps of "no one realized was missing." With no big process document revisions and approvals, the improvement process is very different in Agile. For better or worse- both these situations have drawbacks.

3. **Fail Fast**
   Most teams that are Agile understand the first two points. The third point is often deeper in discussions of Agile—try something. If it works, use it! If it doesn't work,

find out fast and get rid of it! Try something else. This notion of change in process and consistently improving practices and customer satisfaction over *process*, generates a more cooperative, thoughtful, dynamic approach in how-to-get-things-done than any traditional post-mortem can. By definition, agile is dynamic, flexible, and evolving.

4. **People and Interactions Over Process and Tools [Agile Manifesto First Value]**
   Just that! People and our interaction with one another, how we get along and reducing job stress level is more important then what *any* process document says or what some tool dictates as to how you must track, trace and do anything! That is Agile. It is completely *non*-traditional in this sense. It's about people, not process. In post-mortem, the tool and process win. In retrospective— how we get along wins every time!

5. **Role of Project Manager vs. Scrum Master**
   The Scrum Master: the great destroyer of obstacles. The project manager: gets it done. The key distinction between a Scrum Master and a project manager is universally agreed that this scrum master is the destroyer of obstacles. Their primary job is to remove obstacles, to make things easier, get more information, get better estimates, and support the team. A project manager has a very different goal—get the product as close to schedule and on budget.

   If there are tasks that become obstacles, epics instead of user stories, user stories that have bad estimates, insufficient interaction with a customer or business analyst, the scrum master is charged with fixing the problem. If they are not observing these problems themselves or upon hearing it from the team not fixing these problems the scrum master then becomes the problem. Their only job is removing the obstacles. Think about how this impacts the attitude toward change in a retrospective.

6. **Timing**
   Fix a problem when you see it, not six months later!
   Worst case: In traditional projects (not limited to waterfall but waterfall *style* projects) if you spend nine months working on a big project, a week later at a conference table with 20 people you've been arguing with for the last nine months only to continue the frustration in the next project— it's understandable why people are fatigued and angry! When you have a sprint retrospective, the short release/project just ended. You can point out a problem in the user story that came into the backlog three weeks ago and not nine months later.

   Retrospective: immediately fixes and moves on
   Post-mortem: drawn-out and too late to change anything

7. **Amount of Measurement**
   In Agile, measurement is not as consuming as can be in traditional projects. By the book, in Agile you measure burn-down and velocity. That's it. Many teams take some bug measurements, but not the giant dashboards common in traditional projects.

   Traditional projects have more measurement opportunity than Agile projects. Excessive measuring is a time drain and measurement does not always point directly to problems. But efficient measurement can give you a lot of backbone or even evidence for improvement discussions. This is a big bonus in traditional project post-mortems. You may be able to point to evidence to fix problems—this is not the case in Agile. You may only have opinion and feelings.

None of this means that post-mortems are doomed to fail—post-mortems can work! I have many experiences of them being the catalyst for great and successful change. But they are more tectonic. They are best/most successful when there is *mindset change* or paradigm shift. It is tough to have post-mortems work when it means writing or updating a process document, rounds of editing of the process docs and approval and sign-off meetings.

We can continue to do traditional style post-mortems but have them be more effective with a new mindset. You can have an Agile post-mortem! We don't have to call it chickens and pigs, but we can have a more democratic rather than authoritarian process. People can agree to try something and see if it works. Prepare for the post-mortems by highlighting better open communication, searching points of agreement, and remove personal barriers.

It will have a giant effect. We shouldn't get the impression that the grass is always greener on the other side and every Agile retrospective is a resounding success! There are some problems creeping up with companies that have been doing Agile for awhile, especially those who have many Agile teams.

As a result of individual *self-directing* teams doing what makes their work most efficient, they will very often have different practices, different definitions of *done*, differing amounts of unit testing or automated regression tests, tool use, "light, less is more, evolving design" etc. By nature, some teams will say this suggested practice does not work for our team, and changed it. This very often leads to integration problems.

Does this sound like a commercial to become more agile or implement some more agile processes as process improvement? I hope so... ■

SPOTLIGHT INTERVIEW

Mark Levison

Mark Levison has over twenty years experience in the IT industry, working as a developer, manager, technical lead, architect, and consultant. He discovered Agile in 2001 and is now a Certified Scrum Trainer and Agile Coach with Agile Pain Relief Consulting.

Levison has introduced Scrum, Lean and other Agile methods to a number of organizations and coaches from executive level to the individual developer and tester. Levison is also an Agile editor at InfoQ and has written dozens of articles on Agile topics and publishes a blog – Notes from a Tool User. Mark's training benefits from his study and writing on the neuroscience of learning: Learning Best Approaches for Your Brain. Email Mark at mark@agilepainrelief.com.

Offshore testing teams are put between a rock and a hard place. By their very nature what they're being asked to do isn't Agile, at best it's better waterfall. When work is offshored with an Agile team it would be better to have the whole team (developers, testers, ...) in one place.

Much of the magic that happens with Agile Software development comes from building quality into the code and not trying to test in after the fact. In addition, team members start to cross skill with developers learning more about testing and testers pairing with developers to complete feature work.

Much of that is beyond the control of the readers of this magazine. I won't tell you to abandon ship or change jobs. I do suggest it's important to have a good understanding on how this can work so you know what to advocate for. In the future, I think we will see more distributed Agile work and done well. I'm already starting to see teams at some clients with parts of the team in Canada and other parts in India. It works because everybody is working in the same body of code in the same sprint.

Testing is placed at the back of the bus but should instead be an integral part of the process. In addition, the Indians make a real effort to provide an environment where the people are well paid and want to stay with the business, reducing turnover.

1. Specific to testing, how important is it to have a process? Does testing only follow the development process?

In Agile/Scrum there isn't a specific testing process, instead testing is part of every Sprint. The key here is to make sure that test occurs in parallel with development using techniques like Acceptance Test Driven Development (ATDD). Well done testing is really just an ongoing form of requirements elaboration.

2. Concerning "process" on testing projects, would you share with us some common mistakes & lessons learned?

As you see from the above note the most common mistake is to put testers on a separate team and try to test after the fact. The longer we wait between writing a bug/misunderstood feature, the harder it will be to fix. So efforts should first be put into the avoiding the bugs in the first place (writing tests first i.e. ATDD) and then finding bugs within minutes/hours of being written.

3. Do you see a big difference in process from in-house tested projects and outsource tested projects?

As I note above the difference isn't so much between in-house vs. outsourcing as moving from having a combined team to separate teams working a sprint or more later. Assuming you're working in the same cycle, there is still the additional overhead with the reduced communications bandwidth.

4. These days, in 2011, do teams commonly use or follow a formal process guideline, like TMM/TMMi or CMM/CMMi or TPI?

It seems not many companies actually use these. I've not seen any of these in use but that's probably because my clients call me in to help transition them to Agile/Scrum.

5. Many companies call themselves Agile but have done nothing more than shorten the release cycles into sprints, cut documentation and thrown out process. When teams badly implement Agile, how can test teams work to implement team process improvement?

That is rapidly becoming one of the bigger problems in the Agile world, where companies think they know Agile but really its an excuse for cowboy coding. The key in this situation is to use the retrospective to ask questions. Questions that help focus on the underlying problem and not the individuals involved.

6. Sprint retrospectives have become a core practice in the Agile/Scrum world, could you share with us some retrospective techniques to make them most effective?

There are entire books devoted to Retrospectives: Agile Retrospectives: Making Good Teams Great. See also: Agile/Scrum Retrospectives–Tips and Tricks and Agile Retrospectives.

7. We know you are running some Agile testing UK user groups, could you share with us some realistic "best practices" in Agile testing process?

I'm unsure what you mean by realistic, that suggests a limitation I'm not aware of in the abilities of your team members. I've already listed the key areas for improvement when I mentioned development teams including testing, making testing part of the development process and ATDD. As for the "Best Practice" there are none, just good practices in the current context. (There are no Best Practices)

8. Test Measurement/metrics are often used in post-mortem meetings or for process improvement. Agile projects typically have much less measurement. How can we show or justify a need for process improvement with few or no metrics?

Agile Metrics are a difficult problem. There is really only one thing that matters, the high quality business value that we successfully put into use rapidly. That implies many things: bug free, rapid deployment (every 2-4 weeks is a good starting point). Focusing on anything else number of bugs, number of test cases, and percentage of code covered by tests will lead to behaviors that optimize for the measure not the delivery of business value. For more see: Agile Metrics and What is a Good Agile Metric?

9. Sometimes on traditional projects, test teams get a bad reputation for complaining because we are often under the harshest schedule pressure. Do you think this has changed on agile projects? Are our "complaints" now heard as suggestions for improvement?

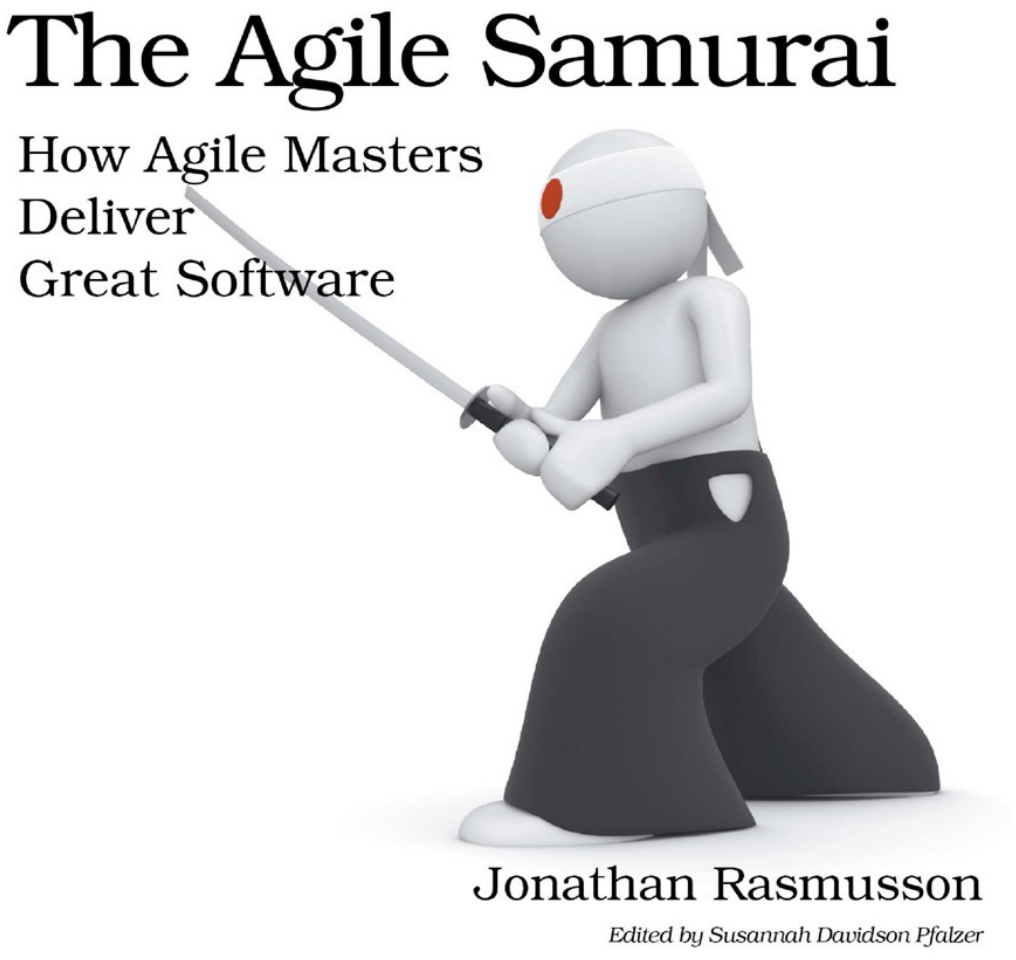
The key here is to use the retrospective as a conduit for issues, don't focus on your problems. Focus on the effect on the customers. Instead of complaining "We were delivered very buggy code on Wednesday of last week" try "We spent alot of time dealing with unexpected problems on Wednesday of last week". Change the focus from specific people to the problem and then ask your team mates to help solve the problem. Remember the goal is for the whole team to succeed in delivering a high quality application to the customer. We succeed in doing that as a team not as individuals.

In the next few weeks I will take some time to write more about the Testing on an Agile team, the item will appear on my blog. ■

BOOK EXCERPT

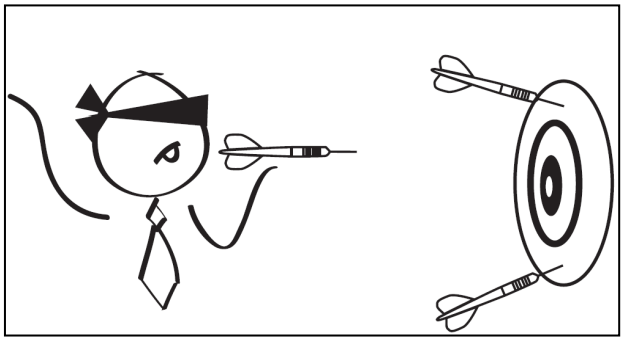
The Agile Warrior

The Pragmatic Programmers



Chapter 7

Estimation: The Fine Art of Guessing



Get ready to bring some reality back to the estimation process. Agile dispenses with the pleasantries and reminds us what our initial highlevel estimates really are—really, they're bad guesses.

But by learning how to estimate the agile way, you'll stop trying to get something your up-front estimates can't give (precision and accuracy) and instead focus on what really matters—building a plan you and your customer can work with and believe in.

In this chapter, you'll learn how to estimate your user stories the agile way, as well as some powerful group estimation techniques for sizing up things.

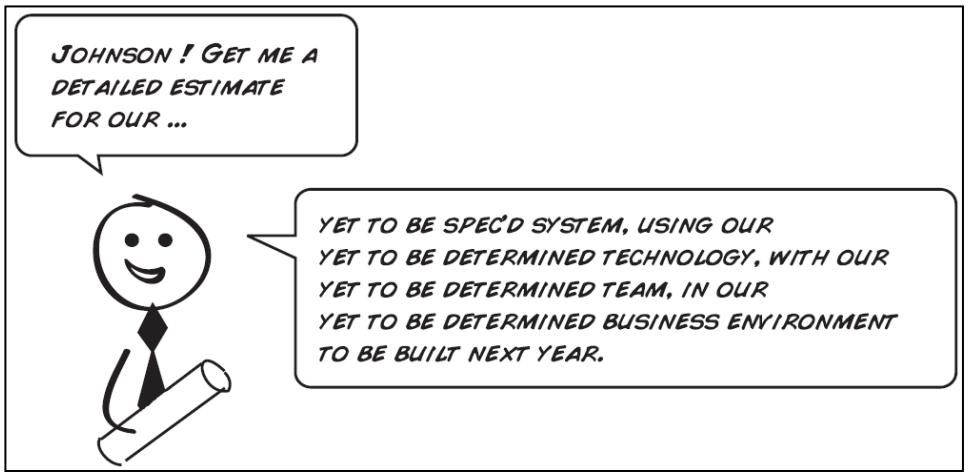
7.1 The Problem with High-Level Estimates

Let's face it. Our industry has had some challenges when it comes to setting expectations around estimates on software projects.

The Point of Estimation

"The primary purpose of software estimation is not to predict a project's outcome; it is to determine whether a project's targets are realistic enough to allow the project to be controlled to meet them."

It's not that our estimates are necessarily wrong (though they almost always are). It's more that people have looked to estimates for something they can never give—an accurate prediction of the future.



It's like somewhere along the way, people lost sight of the fact that

HIGH-LEVEL ESTIMATES ARE GUESSES
(AND USUALLY REALLY BAD, OVERLY OPTIMISTIC ONES AT THAT)

And it is when these up-front, inaccurate, high-level estimates get turned prematurely into hard commitments that most projects get into trouble.

Steve McConnell refers to this dysfunctional behavior as the cone of uncertainty (Figure 7.1, on the following page), which reminds us that initial estimates can vary by as much as 400 percent at the inception phase of our project.

The simple fact is that accurate up-front estimates aren't possible, and we need to stop pretending

they are.

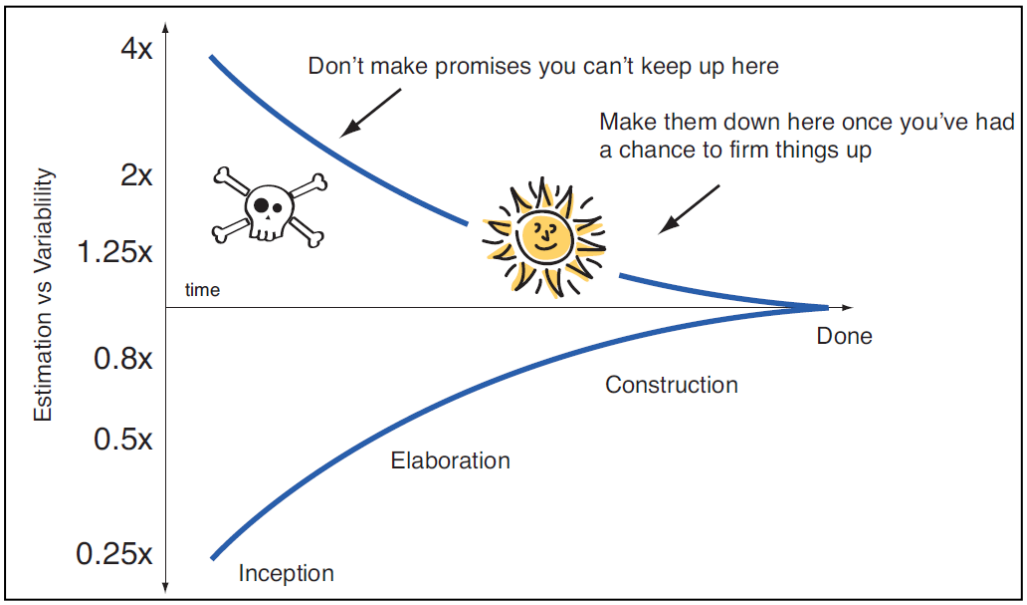


Figure 7.1: The cone of uncertainty reminds us of how greatly our estimates can vary at different stages throughout the project.

The only question our up-front estimates can attempt to answer is this:

What we need is a way of estimating that does the following:

Allows us to plan for the future

Reminds us that our estimates are guesses

Acknowledges the inherent complexities in creating software

7.2 Turning Lemons into Lemonade

In agile, we accept that our initial, high-level estimates aren't to be trusted. However, we also understand that budgets need to be created and expectations need to be set.

To make that happen, the warrior does what anyone would do who is looking to firm up any estimate. They build something, measure how long that takes, and use that for planning going forward.

BOOK REVIEW

How We Test Software at Microsoft

Bryan Pendleton writes an extensive review of the book written by Microsoft's prominent test professionals.

I've been intending to write a book review of *How We Test Software At Microsoft*, by Alan Page, Ken Johnston, and Bj Rollison, but for whatever reason I just never found the time, until now.

In general, I like this book a lot. It's a nice blend of the tactical and the strategic, of the pragmatic and the theoretic, and it covers a lot of ground in a very readable fashion. It's hard to imagine anybody who is seriously interested in software testing who wouldn't find something that interested them in the book. To give you a very high-level overview of the book, here's the "Contents at a Glance":

About Microsoft

1. Software Engineering at Microsoft
2. Software Test Engineers at Microsoft
3. Engineering Life Cycles

About Testing

1. A Practical Approach to Test Case Design
2. Functional Testing Techniques
3. Structural Testing Techniques
4. Analyzing Risk with Code Complexity
5. Model-Based Testing

Test Tools and Systems

1. Managing Bugs and Test Cases
2. Test Automation
3. Non-Functional Testing
4. Other Tools
5. Customer Feedback Systems
6. Testing Software Plus Services

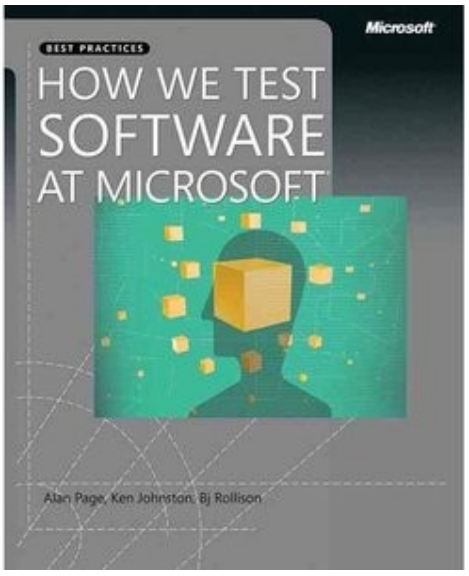
About the Future

1. Solving Tomorrow's Problems Today
2. Building the Future

Now let's take a deeper look at a few of the areas the book covers, in more detail.

Not "those squeegee guys that wash your windows"

The section *Software Test Engineers at Microsoft* describes the organizational approach that Microsoft takes to software testing. I think that Microsoft doesn't get enough credit in areas such as these. Although there are other high-tech companies that are much larger than Microsoft (e.g., IBM, HP, Cisco, Intel) Microsoft is different from these other companies because they are purely a software company (well, OK, they have a very small hardware organization, but it's nothing like the others in that list). I think Microsoft has, far and away, the most sophisticated understanding of how to do very-large-scale software engineering, and I also think that they have one of the most sophisticated approaches to software testing. At a previous job, some of my co-workers did contract



software engineering for Microsoft in their test organization, and it was very interesting to get a peek behind the curtain at how Microsoft works.

The book discusses some of the major tasks and activities that a Microsoft SDET (Software Development Engineer in Test) gets involved with:

- Develop test harness for test execution
- Develop specialty tools for security or performance testing
- Automate API or protocol tests
- Participate in bug bashes
- Find, debug, file, and regress bugs
- Participate in design reviews
- Participate in code reviews

This is a challenging role, and it's pleasing to see Microsoft giving it the respect it deserves.

"The happy path should always pass"

The section *A Practical Approach to Test Case Design* is one of the strongest in the book, and is just jam-packed with useful, hard-won advice for the practical tester. It contains information on:

- Testing patterns
- Test estimation
- Incorporating testing earlier in the development cycle
- Testing strategies
- Testability
- Test specifications
- Positive and negative testing
- Test case design
- Exploratory testing
- Pair testing

It's not an exaggeration to suggest that a professional tester might find it worth getting this book for this section alone, and might well find himself (or herself) returning to re-read this section every year or two just to re-focus and re-center yourself around this level-headed, thorough approach. I particularly enjoy the section's pragmatic assessment:

There isn't a right way or a wrong way to test, and there are certainly no silver bullet techniques that will guarantee great testing. It is critical to take time to understand the component, feature, or application, and design tests based on that understanding drawn from a wide variety of techniques. A strategy of using a variety of test design efforts is much more likely to succeed than is an approach that favors only a few techniques.

"The USB cart of death"

The two sections *Non-Functional Testing* and *Other tools* are also, in my opinion, particularly strong sections, perhaps surprisingly so since they don't at first glance look like they should be as informative as they actually are.

Non-Functional Testing talks about a collection of "ilities" that are challenging to test, and that are often under-tested, particularly since it is hard to test them until relatively late in the process:

Areas defined as non-functional include performance, load, security, reliability, and many others. Non-functional tests are sometimes referred to as behavioral tests or quality tests. A characteristic of non-functional attributes is that direct measurement is generally not possible. Instead, these attributes are gauged by indirect measures such as failure rates to measure reliability or cyclomatic complexity and design review metrics to assess testability.

Here we find a variety of very useful sub-sections, including "How Do You Measure Performance?", "Distributed Stress Architecture", "Eating Our Dogfood", "Testing for Accessibility", and "Security Testing". Sensibly, many of these sections give some critical principles, philosophies, and techniques, and then are filled with references to additional resources for further exploration. For example, the "Security Testing" section mentions four other entire **books** specifically on the subject of software security:

- *Hunting Security Bugs*
- The *How to Break...* series
- *Writing Secure Code*
- *Threat Modeling*

These are quite reasonable suggestions, though I wish they'd included suggestions to read Ross Anderson's *Security Engineering* or some of Bruce Schneier's work.

Other Tools is unexpectedly valuable, given such a worthless section title. This section makes three basic points:

- Use your build lab and your continuous integration systems
- Use your source code management (SCM) system
- Use your available dynamic and static analysis tools

Of course, at my day job we're proud to provide what we think is the best SCM system on the planet, so the topics in this section are close to my heart, but even when I wasn't working for an SCM provider I thought that techniques such as the ones provided in this section are incredibly useful. I've spent years building and using tools to mine information from build tools and CI systems, and I've found great value in static analysis tools like FindBugs for Java; at my day job we're big fans of Valgrind. Lastly, I like the fact that this section observes that "Test Code is Product Code", and so you need to pay attention to design, implementation, and maintenance of your tests, just as you pay attention to these same topics for your product code.

"Two Faces of Review"

Oddly buried in the *About the Future* section at the end of the book is an all-too-short section on code reviews. The section contains some suggestions about how to organize and structure your code reviews, how to divide and allocate reviewing responsibilities among a team, and how to track and observe the results of your code review efforts so that you can continue to improve them. And I particularly liked this observation:

For most people, the primary benefit of review is detecting bugs early. Reviews are, in fact, quite good at this, but they provide another benefit to any team that takes them seriously. Reviews are a fantastic teaching tool for everyone on the team. Developers and testers alike can use the review process to learn about techniques for improving code quality, better design skills, and writing more maintainable code. Conducting code reviews on a regular basis provides an opportunity for everyone involved to learn about diverse and potentially superior methods of coding.

Here, I have to take a brief aside, to relate a short story from work. Recently, my office conducted a two-day internal technical conference. The software development staff all went offsite, and we got together and discussed a variety of topics such as: agile programming, domain-specific languages, cloud computing, and other trendy stuff. But one presentation in particular was perhaps unexpected: our President, Christopher Seiwald, reprised a presentation he's given many-a-time before: The Seven Pillars of Pretty Code. If you've never seen the presentation, give it a try: it's quite interesting. But the important part, I think, is that the company felt strongly enough about the importance of code, and of code review, to get everybody, including the company president, together to spend an hour discussing and debating what makes great code, how to write great code, and so forth.

Is *How We Test Software At Microsoft* a great book? No, it's not. It's too long, and the presentation style bounces around a lot (not surprising for a book with many different authors), and the book is a bit too encyclopedic. I wish that the authors had covered fewer topics, perhaps only one-half to two-thirds of the overall topics, but had covered them in more detail. And the book's unapologetically-Microsoft-only approach can be frustrating for people outside of Microsoft who are interested in how to apply these techniques in other situations: what if you're trying to test low-memory handling on a Linux system, rather than Windows? What should you be thinking about when looking at character set issues in Java? etc. And I think that the book gives rather too much credit to some approaches that I'm not particularly fond of, such as code-complexity measurement tools, model-based test generators, and the love-hate relationship I have with code coverage tools.

But those are fairly minor complaints. If you are a professional tester, or a professional software developer, or even an amateur software developer, you'll find that this book has a lot of ideas, a lot of resources, and a lot of material that you can return to over and over. Don't make it the first software engineering book that you buy, but consider putting it on your personal study list, somewhere. ■

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. Visit his blog at Journal of a Programmer.

2010 GLOBAL TESTING SURVEY RESULTS

Test Process & SDLC

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

Process

The objective of this survey and analysis is to gather information on the actual state-of-the-practice in software testing today. The questions originate from software development team assessments I executed over the years. A process assessment is an observation and questioning of how and what you and your team does.

In such a long survey, I wanted to keep process questions to a minimum and seek free-form comments and opinions on perceptions of process, and save more survey time for assessment of actual practices, strategy and execution rather than an individual's plan for a process. The following section of questions deals directly with the development process itself. Results from this section has always been informative for managers as it deals directly with how things ought to be and what perceptions are for the process itself—in most cases there is a discrepancy between reality and the best laid plans! Let's take a look at the Process portion of the survey.

P-1. How would you describe the effort to release your team's software product or application?

The process is occasionally difficult, but we reach fair compromises to fix problems	43.40%
The projects are smooth with occasional process or product problems	28.90%
It's a repeatable, under-control process.	19.70%
It's difficult; problems every time	6.60%
It's a huge, stressful effort	1.30%

This is an encouraging response! The overwhelming majority, 92% say their development process is either under-control, smooth or occasionally difficult. Only eight percent states the process difficult or stressful.

I am surprised at the very positive response teams have for their development process. This is a very common area for teams to complain and express frustration. Current conventional wisdom has teams frustrated with traditional processes. More than half of the survey respondents self-describe as using development process other than Agile.

P-2. Do you think your SDLC processes are followed effectively?

Yes	52.1%
No	47.9%

Now we see the big disconnect with the first question. This response is a virtual split. Just over half think their SDLCs are followed effectively, and the remaining portion does not.

From my experience, I find in development today is that there are "process docs" detailing how teams should effective make software yet are commonly written outside of development or many times by consultants. Teams regularly disregard these docs and are more comfortable with their own culture or practice that is either expressed or implied in tribal knowledge.

P-3. Have people been trained in the process?

Yes	74.3%
No	25.7%

This is encouraging results matching what I expected for internal training. If a team has a process in place, then it would be easy for the department to create either a PowerPoint slideshow or flash video of the process. Such a tool would make training easy for all employees. The opposite would be a team that does not have a process and works based on either tribal knowledge or "whatever works" ethics—a problematic process for long term goals. Making your SDLC as a training standard is also a great opportunity to question why the team does certain practices and sharpen the process or connect reality to "shoulds."

P-4. During the testing phase of a project, what percentage of time do you spend executing tests on an average week? Example: 10 hours testing in a 40 hour week: 25%

50%	27.30%
74% - 51%	23.40%
Less than 25% of my time is spent on executing test cases	20.80%
49% - 26%	18.20%
More than 75% of my time is spent on executing test cases	10.40%

If there is one big, dark, hidden secret in testing I have chosen as my cause, it is to expose the amount of time testers actually spend testing as opposed to documenting, time in meetings, building systems, data, planning, maintenance—all those things testers need to do as well. The perception of most managers is that testers spend the overwhelming majority of their time executing tests. This is not the case.

Ten percent of respondents say they spend 75% of their time or more testing. This can also be read that only 10% of respondents are testing at least 30 hours in a 40 hour week with 10 hours a week spent on other efforts.

Just over 20% put themselves in the lowest category admitting to less than 25% of their time spent testing. This means 10 hours/week or less is spent testing and during a test phase, spending 30 hours/week working on other initiatives.

Two-thirds, 66.3% of respondents spend half their time or less (20 hours/week or less) in a testing phase actually testing. This is common and always very surprising to managers.

I do not conclude any fault lies in this. I know what test teams have to do to get a good testing job planned, executed and documented. Yet, in some situations, this is an indication of a problem. Perhaps the test team

is forced into too much documentation, must attend too many meetings, or gets called into supporting customers and cannot make up lost testing time.

What is most important here is that everyone concerned—managers and directors, project managers, leads, anyone estimating project size—all know, most testers spend half or less of their time testing.

P-5. How much time do you spend documenting (creating and maintaining) test cases during an average product cycle?

Less than 25% of my time is spent on documenting test cases	51.30%
49% - 26%	27.60%
50%	11.80%
74% - 51%	5.30%
More than 75% of my time is spent documenting test cases	2.60%
We do not document our test cases	1.30%

There are many useful pieces of information to glean from this. Few groups spend too much time documenting. This is a great improvement from just a few years ago when many teams were under tremendous pressure to document every test case. Aside from this being completely useless, it leads to missed bugs! Teams spending so much time documenting were not testing enough causing testers to overlook bugs.

Some teams were collapsing under the stress of regulatory pressure to prove requirements traceability to auditors who were using naïve test case design methods or using a MS Word for test cases.

The last few years have seen an explosion in better test case management tools, better test design methods, like action based testing and Agile methods where lean manufacturing ideas have teams agreeing that less is more when it comes to test case documentation.

Very few teams reported not documenting their tests. This is a significant improvement to just a few years ago; during the dot-com boom when web applications might get a rapid fire test there was no time for documenting any tests, no regression testing, and no repeatability. In the next release, you are expected to start from scratch. All business intelligence is left undocumented and kept with few individuals. Hopefully, those days are gone.

Still almost 20% of the groups report spending 50% or more of their time during a project documenting. That is pretty high. There has to be excellent reason those groups are documenting so much, otherwise this is a problem. If you are managing that time you must ask yourself: do these testers have enough time to test? Is it a test project or a documentation project?

P-6. If your group receives requirement documents prior to the testing planning and test case design process, how would you characterize the adequacy and accuracy of these documents?

Very useful	48.60%
Somewhat useful	48.60%
Useless	2.70%

It is a positive result that only very few respondents find their requirements useless. It is also encouraging to note that almost half of the respondents find their requirements very useful! This is habitually another area where test teams complain of the level of quality of the requirements docs.

An assumption from these results is that requirements docs may be getting better. Perhaps a balance has developed in many teams as to how much information business analysts or marketing teams need to give both developers and test teams for them to do their jobs effectively. That, or, test teams have stopped complaining about requirements and make due in other ways.

P-7. What is your view on the quality of the code that is handed to you at the start of testing?

Usually stable/testable, no idea about unit testing, no accompanying documentation of the build	45.90%
Stable, unit tested	21.60%
Stable with build notes	13.50%
Often unstable with accompanying documentation of known problems	6.80%
Often unstable, little/no information on unexpected changes	6.80%
Very stable, unit tested, with build notes explaining bug fixes and changes	5.40%

To highlight: 40% of respondents appraise their builds as stable; 46% of respondents appraise their builds as usually stable; and 13% found the quality of code often unstable. This all seems pretty good. There is one area that is particularly troubling in all this data.

Almost half of all the respondents do not get information from development. Testers have no idea about unit testing and no information about what changed in the build. There is no viable reason for this and it hurts product quality.

Agile development with TDD and daily scrums is meant to prevent this problematic lack of information. The Continuous Integration practice including automated re-running of the unit tests and a smoke or build acceptance test is very effective in speeding up development and delivery.

The following statements are hand-written responses.

P-8. Please fill-in the blank: My biggest problem with our development process today is:

- "Not all out BA's are trained in writing effective clear requirements."
- "Lack of Unit testing. Unit Testing is NOT automated, very few developers write test harnesses. It is all manual Unit Testing."

- "We have no clue what they are doing."
- "We test our changes, but do not test the overall product.
- Regression testing is our biggest problem."
- "It's a black hole!"
- "On most projects, there is a lack of collaboration and cooperation between test and development teams (these by and large are not Agile projects, of course!)."
- "No technical documentation of what had been build."
- "They are rude with testing team."
- "We need earlier involvement."
- "They don't understand the business or the users well enough."
- " Bad communication."
- " Bad estimation."
- "No timely escalation of probable risks on quality delivery."
- "Too many processes are followed by rote."
- "Bad scope and requirements management"
- "They are laying off QA staff and I'm not sure how they are going to adequately test the product."
- Lots of documentation required that does not increase the quality of the product."

P-9. If you could do anything to make your projects run smoother, what would that be?

- "Better communication."
- "More communication with remote team."
- "More testing by development."
- "Unit testing to be be mandatory & unit test report should be treated as a exit criteria to start the Testing."
- "Send bad developers to training."
- "Spend more time automating regression test cases."
- "Automate our testing."
- "More time allowed for test case preparation / documentation."
- "Re-Focus on planning and requirements gathering. Also we could stand to enforce creation of unit tests by developers. We rely too heavily on QA Automation to catch everything"
- "Get buy-in from key people up-front and work to expose icebergs (blockers to success) as early as possible."
- "Policy in handling customer requests on change requests. Project management and Sales Team have to follow process so as not to over commit on deliveries."
- "Plan to reduce last-minute changes."
- "Lighten the documentation."
- "Stronger project managers that can lead."
- "Better project management, better enforcement of standards for SW development, CM and Testing."
- "Integrate ALM tools."

P-10. If you have a lessons learned, success or failure story about your team's development processes that is interesting or might be helpful to others, please write it below:

- "We have done a good job on creating a repeatable build process. We release once a week to Production. Where we fail is in integration and regression testing."

- "The processes are well-defined. The team's commitment and unity of the team are critical to the success of the project."

- "Don't develop in a vacuum. The less exposure a team has to the business, user needs, how software supports tasks, etc., the less likelihood of success. Get integrated - get informed - get exposed! At a previous company, I used to drag my developers out to customer sites with me and while they dreaded facing customers, they were ALWAYS extraordinarily energized at the end having learned so much and feeling much more connected and *responsible* for satisfying customers. This tactic was ALWAYS valuable for our team."

- "Maintain high morale on the team. Motivate them to learn and develop technical and soft skills." ■

VIET NAM SCOPE

Timeout with Evangeline Mitchell

English instructor Evangeline Mitchell takes a break from her classes to chat about her work at LogiGear VN.

Following highly sought out native-English speakers for teachers in Asia, are Filipinos. They're strong presence in the US medical profession as hardworking and professional employees stand in equal measure within English teaching courses in developing countries. In the Philippines, English is the medium in which all courses are taught from the sciences to the humanities. It is only appropriate that they have the closest accent to an American.

Filipino teacher Evangeline Mitchell has taught English not only in Viet Nam but also in Thailand, Burma, Myanmar and for the United Nations for over twenty years with her American husband, who is also a teacher. Her background is based heavily with international schools and universities but enjoys her switch to a more business environment with LogiGear VN. Mitchell takes time between classes to answer a few questions:

1.Where did you work prior to LogiGear? What's the biggest difference between your last position and LogiGear?
I worked at the American International School. The biggest difference is the fact that I taught English as a second language in an academic setting.

At LogiGear, I help the employees improve their English skills so that they are able to communicate effectively to clients and other English speakers who also work for LogiGear. In a sense, my work has more practical applications as my "students" have already begun their careers and need English in the immediate future.

2. What topics do you discuss in class?
For the advanced communication class, topics can vary. However, whatever the topic, they include useful expressions and idioms that are commonly used in an American workplace. I also use topics as a means to provide opportunities for persuasive speech and debate so that students will be able to express their ideas and points of views in addition to further developing their reasoning skills and vocabulary.

3. How long are your classes?
Classes are comprised of about fifteen students and run for an hour. Pronunciation and Listening Comprehension Practice Classes have a maximum of five students run for half an hour.

4. How many English levels?
For the Pronunciation and Listening Classes , there are 2 levels, Basic and Intermediate. Other classes include Advanced Communication Class, Conversation Class, and Reception Class. Soon, an IT Communication Class will start. All of these classes have only one level.

5. Other than teaching the staff English to better interact with clients, are there benefits to continuing through the English levels?
Absolutely, courses run for twenty hours only. The more the employees are given opportunities to practice and further develop their English skills through attending higher level classes, the more confident they will become as English speakers. So this is definitely a situation where more is better.

6. How else can you interact with LogiGear staff? Do you partake in interviews with candidates?
Yes, I conduct verbal assessments to candidates. The results are given to the HR Department for review. When I am not in the classroom, I generally engage in small talk with current and former students as well as other employees from other departments. This gives the opportunity to use English in a "real" situation that is outside the classroom while also affording the opportunity for students to make practical applications of what they have learned. I encourage them to continue to do this. ■

LOGIGEAR MAGAZINE
MAY 2011 | VOL V | ISSUE 4

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