



# Bach on the future

Prominent tester and author **James Bach** visited the UK and Ireland recently to conduct workshops in his Rapid Testing approach for IT consultants Newell and Budge. We asked for his thinking on some key developments in testing

**PT:** We hope you are enjoying your visit, and meeting testers in Ireland, Scotland and England. What have been the high points so far?

**JB:** The high point has been noticing all the little differences between America and the UK. An important part of testing is imagining other possibilities, so it's good practice to notice how, say, utility signs are designed differently. Things I had assumed are phrased the same way in every English speaking country are not the same at all. I particularly liked the sign outside of Heathrow that said "This sign not in use". That's the first self-denying sign I've seen.

**What are your impressions of testing and software development as done in the UK as compared to the US?**

To me, the UK technical culture seems more concerned about paperwork than are corresponding companies in the USA. The culture over here seems more conservative, to me. The kinds of arguments I hear against exploratory testing, for instance, remind me of what I was hearing 15 years ago, in the States, but relatively rarely today. But then again, I'm part of the American testing counterculture: the context-driven test methodologists. So it could be that there are many companies in my homeland that I would also find conservative, except that those kind of companies don't invite me to work with them. My perceptions may be skewed.

**What is the relationship between Exploratory Testing and Rapid Testing?**

They go together nicely, but they are different things. The terms mean roughly what the dictionary suggests. Rapid testing overlaps with exploratory testing, but it's possible to do test rapidly, yet in a non-exploratory way, and vice versa. Exploratory testing, practically speaking, means simultaneous learning, test design, and test execution. It's the opposite of scripted testing.

Rapid testing emphasises speed. The opposite of rapid is not *slow*, of course, but *thorough*. If I test with perfect thoroughness, I will never be finished, so all real testing is based on some trade-off between rapid and

thorough. Rapid testing, as I think of it, means the most rapid testing that I can do while still being thorough enough for the situation at hand. The approach is getting popular because companies are looking for ways to work smarter in a resource-constrained world. My style of rapid testing is heuristic rather than didactic. That is to say, its structure comes not from standardised activities and prefabricated instructions handed to the tester, but rather by heuristics (guidelines and suggestions) applied by skilled people (or by novices duly supervised) who use them to make situated decisions.

What makes it rapid are practices such as:

- context-driven test planning
- risk-based test design
- risk-focused test cycles
- chartered exploratory testing
- concise documentation
- testability advocacy
- opportunistic test automation

Each of these is a label for a set of skills and behaviours that I teach in my classes and consulting.

**Do you foresee automated tools to support Exploratory Testing? What will these be like?**

They're all around. To begin with, any test automation can be pursued in an exploratory fashion, and that's just how I do it. I write throwaway test programs in Perl or Visual Test for special purposes. Besides outright automation of tests, there are all kinds of useful tools that help me with my ET: automatic logging utilities, program scanners, system diagnostics, etc. Any program that helps me see more clearly or control the software better can be an exploratory testing tool. A good exploratory tester cultivates a knowledge of tools and uses them as the opportunity arises.

Sometimes people who like to program get the idea that they can write something that will do the work of an exploratory tester. That's a futile quest, because thinking of the creative, lateral sort that testers do can't be reduced to an algorithm, except in very narrow domains.

The real motivation behind building such tools, I suspect, is that it's an excuse to write yet another program for those people who have more fun programming than testing. I spent some time trying to figure out a commercial product which was originally designed as an exploratory test tool, and eventually gave up. I couldn't make it do anything for me. Reading the manual, I was also unable to determine what kinds of problems it thought it could find "automatically". I got the impression, by reading between the lines, that it randomly presses buttons looking for crashes. Pretty weak testing, in other words.

**There's been a lot of debate recently about whether there is a role for software testers in agile, test-as-you-code development methodologies such as extreme programming (XP). Can Rapid Testing help with this?**

Agility is about reacting quickly and efficiently to change, and so is rapid testing. There's a deep affinity there. In fact, a group of us in the context-driven circle are in the process of comparing our ideas with those of the agile development people. I'm hosting a workshop, in June, where a group of XP thinkers and testing thinkers will square off and spend eight days developing and testing software together. We call it the Agile Fusion workshop. Our goal is for the testers and the XP guys to educate each other on our various methods.

A challenge for rapid testing is that XP programmers, as a rule, seem to believe they already know how to test. It can therefore be hard to get them interested to learn more about it. Of course, they do know some important things about how to test, but no XP programmer I've yet met has had a systematic approach to testing that would seem reasonably comprehensive to a testing specialist. And most of them seem positively obsessed with the technology of test automation.

**As an ex-programmer, where do you stand on the question of whether having knowledge of software development makes better testers?**

All other things being equal, knowledge of technology absolutely makes a better tester. How could it not? But this doesn't mean all

testers, or even most testers, should necessarily be programmers, too. I think not so much in terms of individual testers, but rather test teams. I think a test team benefits from a diversity of skills and backgrounds.

***It's not often we get to talk to ex-Microsoft testers about Microsoft. What's your opinion of Microsoft's end-user-product quality assurance?***

I never was a Microsoft tester, actually. I worked for Microsoft in the capacity of a consultant and trainer. I designed a formalised exploratory test methodology for use in their compatibility testing program. My brother has been a Microsoft tester in various groups, though, and I have many acquaintances at Microsoft. So, I do have some idea of how things work, there. The most important thing to understand about Microsoft is that their tremendous success has a lot to do with pushing control of processes down to the level of each individual business unit. There is no "Microsoft test process" as a unified and consistently practised thing. Thus, some groups do a great job of it, and some do a poor job.

The other thing to know about Microsoft is that they have an ongoing and awesome challenge with backward compatibility. We used to say at Apple Computer that the only reason God could create the world in 6 days was because He didn't have an installed base.

***Putting testing aside for a moment, for those of us interested in the IT industry in general, your journey from Apple to Microsoft via Borland and HP seems a fascinating one! Which of these organisations do you admire the most, and why?***

I admire Borland the most—in the time before Microsoft hired away the cream of our people. The Borland Languages group of the early to mid 90s was full of fighting spirit and brimming with ideas. That was the proving ground for many of the ideas I talk about today. That was where I experienced how smart and committed people can produce complex software that works very well even though their processes bear little resemblance to the V-model or the waterfall. The thing I remember most about Borland was how every project meeting ended with an appeal for new ideas for how to do our work better.

***It seems to be many people's perception that you and other proponents of the context-sensitive approach feel that trying to develop a generic test process is misguided, except where compliance with regulatory standards is required. Is this correct and, if so, what is your opinion on the current moves towards test process improvement using, for example, the ISO/IEC 15504 (SPICE) model?***

I'm pleased to see that all the hard work I'm doing to create that impression is paying

off. I wrote an article called *The Immaturity of the CMM* in 1994, after all, and many other articles where I have criticised the status quo of process improvement. The problem with the generally prevailing culture of process definition in our industry is that it is applying manufacturing thinking to a social and intellectual process. This is wrong on a very fundamental level. I find that most process people seem to misunderstand the very physics of the thing that they seek to improve. They don't study the social sciences, and don't realise that process improvement is a form of amateur social science research. To learn more about this, read *Cognition in the Wild*, *The Social Life of Information*, or *Things That Make Us Smart*. (These books can be found at [amazon.com](http://amazon.com) – Ed)

I go into a process improvement situation knowing that it is difficult to determine what the real processes are, and that writing down a process changes it in basic ways. I approach the effort with the humility of knowing that I have no power to dictate processes of thinking to other thinkers. I can only inspire, demonstrate, and maybe give a nudge here and there. And I know that any process I try to create can only be experimental until it goes through substantial field testing. Finally, I avoid defining intellectual processes in terms of explicit instructions. It's far more efficient to use training and mentoring to do the heavy lifting, so that the documents we end up writing are nice and thin; reminders to aid thinking, not tutorials or commandments.

So, yes, I think generic testing process documents are either trivial or dangerous, unless they are not intended to be followed. The process work I do is not intended for followers. When I'm dealing with followers, I don't help them with process documents, I help them with personal supervision and guidance.

I believe it's important to challenge people who have the audacity to think they have a good idea about what other people should do. In that spirit, I want to answer some questions which weren't asked in this interview but which are often put to me:

When I say that I don't believe in certification programmes, others often counter that it's an important part of maturing our craft to begin to say something about what testing is and isn't, and what words to use to describe it, and what techniques are generally accepted as useful. I agree, except that there is no "our craft". There is no single community of testers. A certification program is, by definition, certification relative to one particular community of testers who put that program together. What we're seeing is very little communication among these communities, and even when there is, the communication both polite and shallow, or unfriendly and shallow. It's rarely profound and revealing.

The context-driven community is a group of people who frequently get together in small groups to practice discussing methodology. We call ourselves "context-driven" because the realisation that context is king is the first step in building a community that can encompass the full spectrum of testing practice, instead of canonising one set of practices and terminology at the expense of all others.

This will someday allow testing skills certification in a way that is meaningful outside a particular clique of personalities. Today, that is not possible. For the time being, if we are to be intellectually honest, we must admit that there is no social or scientific basis for declaring any terminology or set of practices as something everyone should know or do, everywhere. If we are to be intellectually ambitious, I think we should encourage each other to experiment and learn. Try different things. Violate conventions. Share experiences. That's the fastest road to craft maturity.

Of course, it's a fair question to ask why anyone should listen to me. My standard answer is that I have no reliable credentials. I just do my best to make people feel the better educated for having temporarily lent me their attention. I succeed often enough to remain encouraged. Fortunately, my consulting and teaching approach does not rely on appeals to my own or anyone else's authority. It appeals to the experiences and intelligence of each listener. I also will engage in a conversation about testing with almost anybody at almost any time. I have a reputation for that. In a world of colleagues who shrink from conflict among ideas, maybe that's a kind of credential.

Finally, it's well known that some prominent names in the testing field consider my methods and ideas dangerous. My colleagues and I debate them publicly, and as often as possible. I once had a 90-minute argument with Boris Biezer, under an escalator. I was thrown out of a talk by Tom Gilb for arguing that one cannot measure love (Gilb says you can measure anything). Bob Binder no longer speaks to me. Steve McConnell put me in the classic mistakes section of one of his books because I wrote about the importance of heroism in software development.

There really is a struggle going on for the future of the craft. It's a controversy. We have launched a frontal assault on the orthodoxy, along with the agile development people. We think the future is not in disembodied process descriptions, but rather in super-embodied skill sets and heuristic tools. I don't know if I'm right or wrong, ultimately. All I know is that I can explain, demonstrate, and defend the context-driven approach, and that I am always available to debate it. I aspire to be a precise and scientific thinker. So, my colleagues and I do not hide from criticism, we seek it. We have thrown down the gauntlet. PT