Erik van Veenendaal- Founder - Improve Quality Services Ltd.
March 2013

Forgotten tools

Of course, most often we discuss tools that are by far the most popular, such incident management, test management, configuration management and test execution tools. In this contribution I would like to discuss three types of tools that I have found useful throughout my testing career, but still have a small uptake and are most often not the tools that come to mind first when discussing test tools. I do not intend to provide a full explanation including pro's and con's of these tools, the article is just meant as a reminder to also put them to the front of your mind. I recommend to also consider these tool types when defining a tool strategy, and to not just stick with the more common ones.

Code coverage

Coverage tool: A tool that provides objective measures of what structural elements, e.g. statements, branches, have been exercised by a test suite. [ISTQB]

Having once been a developer myself, I would have loved to have had such a tool back then. As many others, I thought of some test cases (without much clue as to which parts of the code were executed and which parts not), but if the test cases ran ok, I considered the software that I had written to be ok as well. I believe this way of working is still common in many, if not most, development organizations. My first practical experiences with coverage tooling was in a TV project in 1996 using a non-intrusive freeware tool. Developers loved it, it supported them in finding out what part of the software had not yet been covered by the tests on a detailed level. I believe most developers are quality-minded like us, but we need to provide them with the knowledge and supporting tools to be able to deliver quality. Coverage tools do exactly that. Of course they can also be used to define strict and measurable exit criteria for component testing. Beware, if you go too strict too soon, otherwise resistance will become an issue. Finally, these tools can also be used for continuous integration when having an automated test suite that runs overnight. We can very easily track the quality of the test suite over time by measuring its coverage (see figure 1.) Nevertheless, recent surveys [PT] show that not even 10% of the development organizations are using a coverage tool. However, with the strong focus

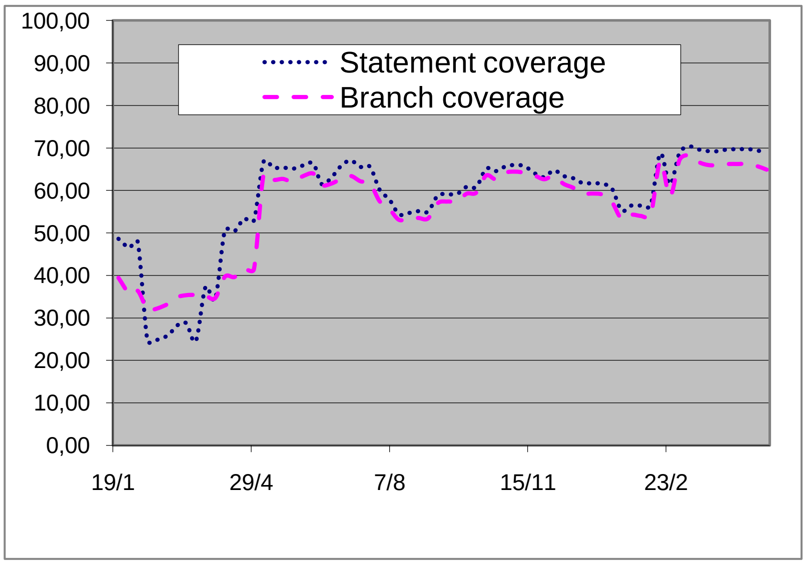


Figure 1: Coverage measurements to guide test improvements

on component testing within Agile development methodologies this should change rapidly.

Static Analysis

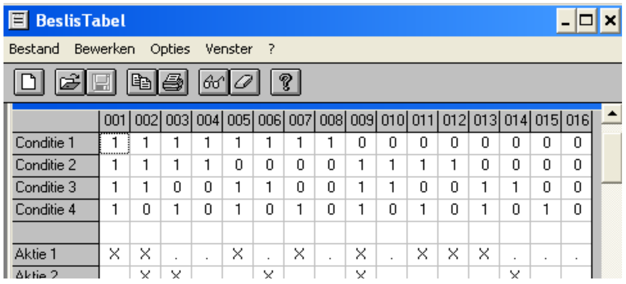
Static code analyzer: A tool that carries out static code analysis. The tool checks source code, for certain properties, such as conformance to coding standards, quality metrics or data flow anomalies. [ISTQB]

Many of the arguments that I mentioned when discussing code coverage tools also apply to static analysis tooling. Again, if used in the right way a highly useful tool to support a developer in producing quality software. However, most organization implement static analysis tooling at an organizational level. This may be the preferred situation (according to theory and vendors), but is not always a feasible one. Organizations then end up in endless discussion to get full agreement between developers on style guides, coding standards etc. And what about applying the new style guide and coding standards retrospectively to all the legacy software that is already in place and will be there for at least the next decade? Not without reason is static analysis in the top 4 for shelfware tooling [PT]. If implementing it in full on an organizational level is asking too much, don't do it!! However, this does not mean that static analysis tools cannot have added value. Perhaps we should keep it much more simple, focus on the twenty or so coding rules we all agree on. Define a minimum set of software metrics with criteria we all agree on such as cyclomatic complexity, number of nested levels and comment frequency, and provide the tool to the developers to start using. I have seen great results in producing more maintainable and more reliable software by applying static analysis tools in a just a limited way. Perhaps the 80/20 also applies here. Remember research has taught us that 40% percent of the failures in the field could have been prevented if static analysis was used. In practice important quality attributes such maintainability and reliability are often forgotten in Agile projects; a static analysis tool that provides support in checking for compliance with the most critical coding rules and software metrics will have added value here as well.

Test design

Test design tool: A tool that supports the test design activity by generating test inputs from a specification that may be held in a CASE tool repository, e.g. requirements management tool, from specified test conditions held in the tool itself, or from code. [ISTQB]

In many testing courses much attention is given to test design techniques, including exploratory testing. Of course it is important to teach people how to design test cases, to some extent it's the heart of testing. However, recently I read a survey stating that approximately only 50% of the testers explicitly apply test design techniques and around 25% percent apply more than one technique. (Food for thought!) In nearly every test design technique there are steps that would benefit from being, at least partly, automated. Most large test tool providers seem to have no idea what test design techniques are and would benefit largely from an ISTQB Foundation Level course. As a result there is still limited availability regarding test design tools,



	001	002	003	004	005	006	007	008	009	010	011	012	013	014	015	016
Conditie 1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
Conditie 2	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
Conditie 3	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
Conditie 4	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
Aktie 1	X	X	.	.	X	.	X	.	X	.	X	X	X	.	.	.
Aktie 2		X	X			X			X					X		

Figure 2: Decision table tool screenshot

hence the low uptake. This is a paradox since we perceive it as being a very important part of testing, but tool focus is on test execution, test management and incident management tooling. However, if you start searching you will come across all kinds of easy-to-use tools that support test design techniques. These are not professional tools (there are one or two exceptions), but tools developed by someone enthusiastic about testing. In The Netherlands we have a tool called BTWIN which is in fact not more than an advanced Excel sheet, but does support decision table testing (including collapsed tables) perfectly (figure 2). I'm also using a small tool that supports me whenever I have a difficult set of (business) rules that require testing using condition determination coverage; many of the readers are probably familiar with the freeware CTE XL tool that supports classification trees (figure 3), etc. None of these are spectacular tools, but they should be in every tester's workbench as they make the application of test design techniques easier and thus eventually will lead to a larger uptake.

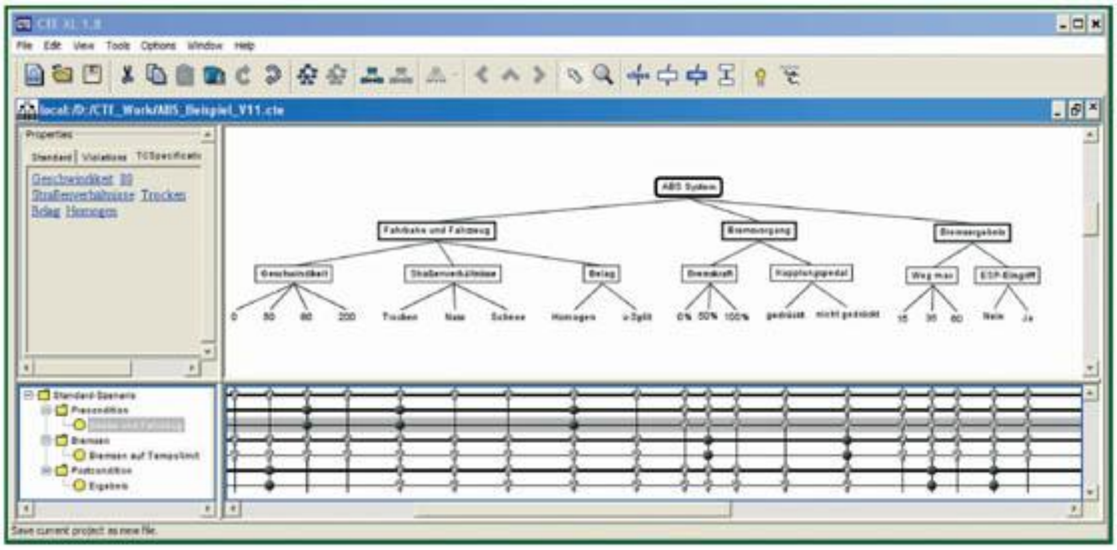


Figure 3: Classification Tree Editor screenshot

Individuals over Processes

It was only when writing this column it struck me that I was making a case for simple easy-to-use tools over full-blown professional tools. Don't get me wrong, full-blown professional tools offer great support but sometimes there are alternatives depending on the maturity level and domain of the organization, development processes in use etc. In fact in my opinion a test tool strategy can be a combination of both, one doesn't exclude the other. Providing engineers (developers/testers) with a personal tool workbench consisting of easy-to-use and highly supporting tools allows you to get the best out of people. To some extent I'm re-stating "Individuals over processes". Does that sound familiar?

[ISTQB] E. van Veenendaal (ed.) (2010), *Standard Glossary of Terms Used in Software Testing Version 2.1*, International Software Testing Qualifications Board

[PT] E. Van Veenendaal, Tools and the last six years, in: *Professional Tester*, November 2010

For more information

Erik van Veenendaal (www.erikvanveenendaal.nl) is a leading international consultant and trainer, and a widely recognized expert in the area of software testing and quality management with over 20 years of practical testing experiences. He is the founder of Improve Quality Services Ltd. (www.improveqs.nl). In 2007 he received the European Testing Excellence Award for his contribution to the testing profession over the years. He has been working as a test manager and consultant in various domains for more than 20 years. He has written numerous papers and a number of books, including "The Little TMMi", "ISTQB Foundations of Software Testing" and "Testing according to TMap" and recently published "Test Maturity Model integration – Guidelines for Test Process Improvement". Erik is also a former part-time senior lecturer at the Eindhoven University of Technology, vice-president of the International Software Testing Qualifications Board (2005–2009) and currently board member of the TMMi Foundation.

Erik Van Veenendaal – Founder – Improve Quality Services Ltd.
eve@improveqs.nl
http://www.erikvanveenedaal.nl

follow Erik on twitter @ErikvVeenendaal