# The Homegrown Tools Syndrome

By Michael Stahl Jun 2012

This article was first published on www.stickyminds.com “Weekly column” on August 2012 ( <http://www.stickyminds.com/s.asp?F=S17601_COL_2> )

Summary: Test management is a generic process, yet much effort goes into developing tools in house to do this work. Learn the reasons for this phenomenon and suggestions for avoiding it.

Test management is a rather generic process. The basic building blocks are the same: test cases, test suites, test cycles. How much different could it be from one team to another, or even among different companies? And yet, I have seen much effort spent developing tools in house to do this work.

Why does this happen? Why spend time and effort reinventing the wheel, when this wheel can be brought home from open source or commercial options?

And, it’s not just test management. I’ve witnessed this happening with other generic tools, such as test execution engines, bug tracking, and requirements management tools.

To me, the fact that the phenomenon happens across a wide range of tools is an indication that the reasons are predominantly non-technical. It’s not that test management is an art that is not yet fully understood by commercial tool developers and therefore has to be designed in house. There are other reasons at play that cause this seemingly redundant effort.

In this article, I propose a number of reasons that I think lead to internal tool development. While I refer to “test management tools” (TMTs), the logic applies to a number of other tool types.

**"We Are Special"**

The common wisdom when discussing tool selection is "Find a tool that supports your process." This is one of the main reasons why teams end up developing their own tools.

How's that?

Commercial TMTs are developed with a certain process in mind. They will support this process best. They *can* support some variations of the process, but when attempts are made to use them to implement a process significantly different from the one they were designed for, the tools are limited, counter-intuitive, and annoy the users.

For most teams, the process is already in place when someone decides to shop for a tool. The team is almost guaranteed not to find a tool that supports their process exactly. It is, therefore, very easy to conclude that "none of the tools out there support our process" and that the team will be better off developing its own tool. This flawed logic assumes your situation is unique and therefore you have special needs from a TMT.

The motto is "We are special." Here is a flash: You are not.

It may be good for team morale to think of yourself as special, but your process isn’t different from other team’ processes because your project or technology is so special. It's just that the “process" is a natural result of technology, knowledge, personalities, and history, all lumped together into "how we operate." Replace your team with a completely new team of managers and engineers, and they will end up with a different process for the same situation. Any of these solutions can be as good as the others.

With the above in mind, my advice when looking for a tool is the following:

1. Select a tool whose underlying process is *close to your process*.
2. Then, *modify your process* to fit the tool.

Adopting this suggestion will avoid a situation I have seen a number of times: A team selects a tool and then "fights" the tool to force it to support existing process. This is done by writing customization code and using the tool in ways that are not intuitive to the user and that disrupt the underlying flow built into the tool by its designers. Some of the tool's capabilities are not used at all, since they don't fit into the existing process. The result is usually a frustrating user experience and never-ending work maintaining and updating the customizations.

**Natural Evolution**

Another reason I see for the development of a TMT in house is natural evolution. It's not an easy or natural call for a new test team to buy a TMT. In startups or small test teams within a large corporation, the normal evolution of a test team is to start small—a tester or two—and slowly grow as management realizes that testing is needed. Often, the growth follows the development team’s size, which also starts small.

For a small team running its first project with no legacy to support, a TMT seems like overkill. It's much simpler just to have the test cases in Excel and report results manually. As the project gets more complicated and more people come on board, there is a need to expand the capabilities of the simple spreadsheet. In many cases, this takes the form of Excel macros or small pieces of code that extract and report data. Coding these capabilities is a "side job" of a tester with background in programming. It's an ad-hoc solution never meant to scale to a large system.

Time passes, and the test project becomes more complex. New versions are released, while older versions need to be maintained. More testers are added to the team. User support takes more time, and new feature requests accumulate. It becomes clear that a more serious effort is needed. The tester who owned the macros is now managing scripts and macros full time.

More code is added, aimed at matching exactly a process that evolves naturally with the team and the projects challenges. By this time, there’s a significant investment in the scripts, code, and macros. Even when the system starts to burst at the seams, it's a hard management decision to say, "Let's dump all this work and get a commercial system." Too much emotional attachment and professional pride is attached to the system. On top of that, the test manager needs to go to higher management and request budget to replace a system that he previously presented as one of his team's highest achievements.

All of this works against the decision to get a commercial system, thereby strengthening the in-house system’s hold.

**Hubris**

Another reason why teams adopt homegrown tools is inside pressure by test engineers who would like to develop such a tool and do not have a good grasp of the complexity of the problem. The exact situation varies. Sometimes, there is an existing homegrown tool and the question is whether to replace it. Other times, there is nothing but Excel sheets and the need for a strong test management tool is clear.

In these cases, management has already concluded that something needs to be done and is ready to spend the money. But, someone on the test team makes a strong argument against buying a system with the promise that “we can build a better system, perfectly fitting our needs, faster and cheaper than a commercial solution.”

In other words, “While there are whole companies with a large number of developers whose core business is to develop test management systems and who have put years of knowledge into their code, we can do it better and in about three weeks.”

It’s very easy for management to fall for this. There seems to be little to lose. At worst, it postpones the purchase decision by the three to four weeks it takes the team to work on something.

The problem is that the resulting something is imperfect—a bit of a kludge—that seems to do most of the stuff it should. It makes sense to delay the purchase decision a few more weeks, and we are back to the “natural evolution” situation.

**Fun**

Woven into all the previous reasoning is the human angle. Developing a test-automation framework can be more fun and challenging than running regression tests. The source of many grassroots initiatives of test automation is a tester with programming skills having fun developing a small tool or script. This, as outlined above, may later mushroom into a full-fledged automation framework. The tester-developer will not want to give up the professional challenge and the satisfaction gained from automation development. The developer will point to the fact that “our needs are not supported by external tools” (we are special) and that “I can do it cheaper and faster” (hubris).

In addition to the reasons mentioned in the “evolution” section, the wish of management to keep the tester-developer happy also weighs favorably towards continuing the project. Making a management decision to abandon internal development may mean loss of talent, as the frustrated tester-developers decide to leave the team.

**What’s to Be Done?**

Words are cheap, while the emotions and financial considerations of moving to a commercial TMT are not trivial. With this humbling thought in mind, I will offer a few ideas.

I already suggested how to overcome the “we are special” syndrome: Select a tool that closely match your process, and modify your process to fit the tool. But, the other issues are predominantly non-technical and are difficult to solve because they are so tightly tied to the human side of the business—the professional aspirations, self-fulfillment, and morale of specific people whom the organization wants to keep.

The “natural evolution” problem can be improved by management awareness. Since you know this is how things will end up, be alert when a small, local solution starts to build up to a large, poorly designed system. (Read more on this in “[Patterns in Test Automation Project Failure—and Recovery](http://bit.ly/Patterns_in_Test_Automation_project_failure-and_Recovery)” [PDF].)

When encountering a proposal that seems too good to be true, you may be experiencing hubris firsthand. One way to deal with it is to ask the people making the proposal to list the requirements they plan to implement. Read the list, looking for ambiguities, missing details, and incorrect assumptions. In short, thoroughly review the proposal as you would review requirements for a product you are responsible for testing. It may also be worthwhile to ask that the proposal include a definition of what will be considered a failure—i.e., when we will admit that we did not deliver and the project should be abandoned.

This will achieve a number of goals. You, the manager, will know what you are getting (and what you are *not* getting). The team making the proposal may get a reality check and understand that they oversimplified the problem. A side benefit of this exercise is that the requirements list can be used in the evaluation of candidate commercial tools. Additionally, the definition of “failure” allows you to later cancel the project, based on what the team defined as valid reason to do so.

The last one—the “fun” factor—is probably the hardest one to solve. You need to tell engineers to stop doing something they draw great satisfaction from. You need to decline their project proposals. You are indirectly telling them that their ideas are wrong and their time is better used doing something they don’t particularly look forward to.

Welcome to management.

Sometimes, you have to make the hard calls and face the resulting personnel difficulties. Being a great manager means knowing how to identify opportunities that leverage employees’ strengths and let them pursue their own interests. But, another trait of a great manager is being courageous enough to put your foot down and kill a project when you know that project is not the correct way to go.

A lot of grief can be avoided by understanding how *not* to get to the point where people are heavily invested (professionally and emotionally) in a tool that should not have been started in the first place.

**Conclusion**

There are a number of reasons why teams end up developing their own TMTs. Most of these reasons are wrong. While there may be some unique cases that justify the effort, in most cases it makes more business sense to adopt an existing open source or commercial tool. Instead of re-solving an already solved problem (test management), challenge your tester-developers to solve test problems directly related to your business and technology.

No external tool will solve *that* for you.

**Acknowledgements**

[Dan Alloun](http://www.softwaretestingclub.com/profile/DanAlloun) and [Shmuel Gershon](http://testing.gershon.info/) reviewed the draft of this paper, and their valuable advice improved it significantly.

The idea to “ask what will be considered a failure” came from [Mooly Eden](http://www.linkedin.com/pub/mooly-eden/18/a9/ab4), Intel VP.