BUILDING A CODE BONEYARD

BY MIKE KELLY

Great testers suffer from overproduction – they generate more ideas than they could ever reasonably use. It happens so often, we have testing techniques and approaches that are focused on enabling you to make decisions around which tests you should run (because you can't run them all) and which tests you shouldn't. If you are new to testing and don't yet suffer the pain of overproduction, we also have test techniques and approaches that will help you come up with more ideas for testing than you could possibly use.

Why does this dynamic exist? Why would we on one hand have techniques designed to help us generate more ideas than we could possibly use and on the other hand have techniques that help us narrow the scope of our testing?

One problem testers face is that we don't know what we don't know. This causes us to error on the side of producing too much in an effort to compensate for the unknown; and as part of that idea production process we learn about what it is we are testing. Once we sufficiently understand the problem, or think we sufficiently understand the problem because we've produced as many ideas as we possibly can, we can then make informed decisions around which tests might be the right tests to run. This is one of the main reasons overproduction occurs.

People who are experts at producing ideas have a wealth of ideas, data, and experience available to them; more than could possibly be required. Testers who are good at overproduction make idea production cheap, quick, and diversified so they don't worry about getting it right the first time. Any one of their ideas can be a bad idea or could miss the mark and that's ok because they know they will get at least one idea right and will abandon the ones that don't work. This technique is commonly referred to as "shotgunning". Another familiar example of overproduction is the classic brainstorm. Neither of these activities are wasteful if the effort is relatively inexpensive and the ideas sufficiently diversified.

Overproduction often results in growth of the tester; having produced something once you can more easily produce it again, making you more skilled as a result of overproduction. For example, each time I need to produce ideas for performance testing I get a little bit faster and my list grows a little longer. That's not accidental. It's because I'm systematic in how I produce my ideas and I'm systematic about how I abandon and recover them.

Abandonment is another key part of idea generation – without it testers would be completely overwhelmed. A tester who is good at abandonment knows when to stop an activity or leave

data behind and to move on with other activities instead. Ideas that are no longer useful should be abandoned in a way that allows them to be recovered later when a context in which they might be useful emerges. This option for later recovering abandoned ideas produces the ability to refactor an idea, using some or all of the original idea or artifact in a useful way. To be really good at abandonment and recovery, you need to reduce or eliminate the maintenance costs of keeping your ideas at your fingertips.

At this point, you might be asking yourself, "What does all this have to do with code; and where's that cool boneyard I was promised in the title to this article?" Creating a code boneyard is an abandonment and recovery technique. It gives you the ability to freely discard code because you'll know where to look for it later. Overtime, the bits of code (or bones) you discard will start to accumulate and you'll find you have a rich set of examples to draw on when you need to generate

Tactics for managing your ideas

Overproduction, abandonment, and recovery are tactics for how to manage your ideas:

- Overproducing ideas for better selection. Producing many different speculative ideas and making speculative experiments, more than you can elaborate upon in the time you have. Examples are brainstorming, trial and error, genetic algorithms, free market dynamics.
- Abandoning ideas for faster progress. Letting go of some ideas in order to focus and make progress with other ones.
- Recovering or reusing ideas. Revisiting your old ideas, models, questions or conjectures; or discovering ideas already made by someone else.

You can find more skills and tactics critical to the professional exploration of technology here: http://www.satisfice.com/articles/et-dynamics.pdf

code quickly.

I currently have three separate code boneyards that I work with. At work my team has a simple SharePoint list where we can upload scripts (Ruby, SQL, VB, etc...) that we think others might find useful. I also carry a thumb drive with me where I keep a large pile of Ruby scripts and code snippets (the thumb drive keeps my boneyard mobile). And finally, in my webmail account I have a folder for code snippets from mailing lists where I may see something that I can't apply immediately, but I know I might have a use for later.

Here is what's common between those three boneyards:

- Each "pile" has a specific purpose (work code, personal code, community code that might be useful at some point, but I haven't really researched or used yet).
- You have access to it wherever you go. I can access SharePoint anywhere on the office network, the thumb drive is always with me, and I can hit webmail anywhere I have an internet connection.
- You can search it (thumb drive), index it (SharePoint), or sort it (webmail). That is, you can use tools to aid in rapid recovery.
- Here's what you don't want to think about as you build your boneyard:
- Don't think about maintaining the code or worry about compatibility issues.
- Don't limit your boneyard to working code - there can be value in storing ideas that didn't work. Many times your best code can be found in bones created as you struggled with a difficult problem you never managed to completely solve.
- Don't think you always have to go back to the boneyard. I used to keep reusing the same bit of Ruby code that loaded all the filenames in a directory (and all it's subdirectories) into an array. One day I discovered I could actually remember the code enough to write it from scratch.

A code boneyard is more than a dumping ground for physical resources like code that you can abandon and recover. The very act of going through the process of idea generation and pruning changes the tester - even if you abandon an artifact, you still retain in your mind the experiences of creating it. You retain the learning experience, the effort invested in improving your position on the learning curve. The next time you need to create something similar you are better at it and you will be able to achieve the results you want more quickly.

About the Author

Mike Kelly is currently a Software Development Manager for a Fortune 100 company. Mike also writes and speaks about topics in software testing. He is currently the President for the Association for Software Testing and is a co-founder of the Indianapolis Workshops on Software Testing, a series of ongoing meetings on topics in software testing, and a co-host of the Workshop on Open Certification for Software Testers. You can find most of his articles and blog on his website *www.MichaelDKelly.com*.