Omega Tester: Testing with a Team of One

By James Bach

In the movie Omega Man, Charleton Heston played a scientist-warrior left alive, and alone, after a plague kills everybody on the planet. Well, he's not totally alone. He also has to fight a lot of mutant zombies.

That brings me to testing. Whether it's cost cutting or a headlong rush to "agile" development, a lot of test teams are understaffed, these days. Let me put it this way, if your test team is not understaffed then it soon will be. Many of the testers I meet are the only tester on their projects. Like Charleton Heston in the movie, the "omega tester" must be canny and resourceful. Hey, and even on larger teams, your teammates may be working in different areas of the product, or you might be spread out all over the world. You may feel alone, even on a big project.

Here are the general issues with being an omega tester:

1. **Anything you do means something else won't get done.** (Don't tell me you're going to "multi-task." Multi-tasking is for managers and other people who don't do anything mentally demanding.) The moment you sit down and really focus on testing something, every other part of the product, every communication, every other preparation or learning must be put on hold. While you are testing, you are a single-threaded process, and the queue grows quickly while you're head is down in your work.
2. **You feel too busy to plan and prepare.** There may be manuals to read, developers and users to interview, test tools and virtual test platforms to configure. You need to learn enough about the product to imagine the risks and get a picture of your test strategy. On top of that , there are lots of meetings you could and probably should go to. But as soon as the product is in your hands, the pressure will be on you to test now, now now! "Hey wait," you say, "I'm on a Scrum project, we take it one sprint at a time." Yes, it would be nice if your test preparation activities mapped to the development process. But there's no particular reason that they should. Testing and development are not symmetrical activities. A small amount of development can create a huge test problem. For instance, imagine that a developer slaps together a database and sends it to you for testing. These days that's not a lot of work, for HIM. For you, welcome to the House Of Pain. You immediately have the problem of creating a potentially large and sophisticated test data set. Sure, you can skip that step and just add a few records to test with. That would be a start. But tell me: in what future sprint will your job be to do the deep data testing? They will always add something new, and you will always be expected to jump on it. Developers create functionality, but they also automatically create performance problems, scalability problems, reliability problems, usability problems, etc. They create these problems for free (I've never seen "create bugs" on a development schedule, yet there always seem to be enough time to create them), but it isn't free for you to find them.
3. **You feel too busy to mind your infrastructure.** As you test, you learn. You create files and take crude notes. Your desktop becomes littered with these things. This represents a kind of technical debt that grows harder to support as the project moves along. You need time to stop testing and organize your notes, consolidate and organize your test data, create a coverage outline or update your risk lists. A good infrastructure for your test

project gives you efficiency, but you may feel that efficiency is a luxury you can't afford. Indeed, how many fire-fighters, in the midst of a disaster, decide to change the oil in the fire truck?

4. **You feel too busy for curiosity.** Testing is a search process. Not only that, it's a heuristic search. That means there is no sure way to find what we are looking for, and no optimal way of searching for it. There are certain obvious kinds of testing to do, but that's not enough for excellent testing. Excellent testing requires that you test more than the obvious things in the obvious ways. You must indulge your curiosity. You must play. You must open yourself to the unexpected. For many testers, this can be hard to justify (a lot of my teaching and coaching focuses on solving that problem). It looks like you are goofing off, instead of doing important work. So, when the pressure is on, curiosity is easy to put on the shelf. You then become a little more like a fancy robot; a little less like a powerful human tester.
5. **No one understands what you do.** Programmers and testers both do work that is intangible. The difference is that programmers have a tangible finale: the software. The end product of testing is less tangible, and efforts to make it more tangible mean spending less time actually testing. Many people think testing is easy ("just try it and see if it works!") and they wonder why it takes you so long to find those "obvious" bugs. At least if you are in a team of testers, you can commiserate and share the burden of explaining the impossibility of complete testing for the 37th time. As an omega tester, you just feel surrounded by monsters.

What to do about this? My omega tester suggestions consist of technical ideas and social ideas.

Technical Ideas:

- **Forget test cases and scripts. Think test activities and the testing story.** I don't think of myself as creating test cases when I work, I think of myself as doing testing. Sometimes that results in specific documentation or some other sort of test artifact, but not necessarily. Test cases are not testing. Configuring the product, operating it, observing it, and evaluating it-- that's testing. Because you're overloaded, I suggest thinking about your strategy in terms of what you need to do, or what you need to create in order to do it. A detailed documented test procedure is usually neither of those things. Focus on the activity of testing, or think about the test report you are trying to produce (it may be an oral report), and you'll get more testing done.
- **Let risk be your compass.** A common test strategy practice is to organize testing around a standard, such as a specification or requirements document. Those things are important, of course, but as an organizing principle, I suggest risk, instead. By that I mean focusing your testing on the more important product areas and looking for the more important kinds of bugs. At first you may not have a strong idea about what the risks are, but keep asking yourself "what are the things we are most worried about NOW?" and as the project goes on you will learn this better.
- **Test in short, sharp, uninterrupted sessions.** Session-based testing means that you set aside a block of time-- I prefer 90-minute blocks-- put a sign up at your desk that says "Gone Bug Finding" or something like that, pick a risk or an area of the product, and test it. Record your testing charter (a sentence or two about the objective of the session), too. This is important because as a lone tester you need to be more definite about the progress

you make. It's too easy to let email or meetings drain all your time away and find you've gone a whole day without really testing anything.

- **Use unexpected lulls for infrastructure, preparation, or re-assessment.** These lulls will occur more or less regularly on any project. I'm talking about those times when you're waiting for a new build because they can't get the thing installed properly on your test server, or any other time when your testing is blocked. You need to take those opportunities to look over your strategy, try to think ahead, or pull your notes and files together into a better order.
- **Use concise test documentation.** I'm talking about lists, outlines, or tables. I like to use a wiki for this, these days (I wrote this using SocialText). Or Google Docs works, too. Something online, shareable, that has a way to roll back history. When I say concise I mean write what you need and no more than that. Think tables and lists and outlines. Avoid writing detailed instructions. Don't write things that are easily handled by training. Unless you are testing Microsoft Word, writing is not testing, and for darn sure formatting is not testing.
- **Use recording tools to shorten bug investigation time.** They say that scripted testing is better than exploratory testing when it comes to bug investigation. This is completely untrue. Having a list of steps that you produced two months ago is no more helpful than a list you made as you went along, except you are two months wiser, today, and the notes you record today are fresh in your mind. Of course, even better is if you don't write down your steps at all-- because they were captured automatically by your recording tool. There are various commercial and open-source tools you can use for this. I like BBTestAssistant, TestExplorer, or Spector. But there's also Wink, which is free. Also remember there are log files, especially if you are testing something online. Or use a digital camera.
- **Do lots of exploratory scenario testing.** This means that you test according to realistic and complex sequences of user behavior organized around a coherent scenario. The major downside of it is that to do it well requires that you get real users to help with it or that you have a deep understanding of real users, how they think, and what they will do with your product. Exploratory scenario testing would be the same as beta testing if beta testing were highly organized, focused on challenging situations rather then just every day situations, and done by people passionate to find bugs.

Now for the social ideas. Unlike Charleton Heston, who was being chased by zombies most of the time in that movie, you need social skills to be a really good omega tester. It's your social skills that will save you.

Social Ideas:

- **Declare high service aspirations.** For instance, when I work with a development team, I give them a one page declaration of all the things I promise to do for them. It documents twelve commitments I make to them as a tester. I don't want them to see me as a nagging burden. I don't want to be a buzzing horsefly around their ears. I want to ease their burden, not add to it. I will serve my team in any way I can to achieve that. By making my commitment clear, I find that I foster a similar attitude in return. That makes my job easier, because their help is vital for me to do my job well. The success of any other idea, below, begins with this one.

- **Set low technical expectations.** "Yay, we hired a tester! Now we'll ship without bugs!" People will think testing is easy. People will think your job is to "ensure quality." You should challenge that thinking right from the start. Several articles could be written about how to do that. It will have to suffice for me to say that there is no theory of testing and no tool of testing capable of demonstrating that a product is bug free. There is no set of such tools, theories, techniques that can do that. There is no way *at all* to do that. All you can promise is to look for problems in thoughtful, reasonable ways. And forget about ensuring quality. You support the team in producing a good product. You are part of the process of making a good product, but you are not the one who controls quality. The chief executive of the project is the only one who can claim to have that power (and he doesn't really control quality either, but don't tell him that).
- **Co-locate.** It may not be in your power to huddle around the same desks as the programmers, but baby, get as close as you can. It makes a profound difference in your productivity if you are within ear and eye range of the people creating the product. My feeling is that is doubles my productivity, at least, to work ten feet away versus ten miles away, and quadruples it versus being ten time zones away.
- **Practice, practice, practice explaining testing.** It's not enough to know how to test. If you are in a team of one, congratulations, you now have to be ready to explain testing, defend testing, and teach testing. Over and over. To the same people, too, because they will keep forgetting what you tell them. One way to practice is to blog about testing, or write about it in a forum like the Context-Driven forum (software-testing@yahoogroups.com).
- **Advocate for testability.** Testing and development are not symmetrical. A small decision made by a designer, making little work for him, can be a whole lotta work for you. You've got to get in there, gently, and help them understand the testing impact of what they add or change. Don't expect anyone to know about testability unless you tell them what that is. Basically, it boils down to two big things: A) can you see it? B) can you control it? That's why I push for rich log files and a scriptable interface (I'm happy to test through a GUI, but I also want to get underneath the GUI to control the product better and see it better).
- **Advocate for strong pre-testing.** I don't worry about this so much when I'm working with a good test team. It can be hard to get developers to work methodically. It is especially hard to push for that without sounding like a nag. But if you are a one-person team, you don't have much choice. You need to sit down and level with them: unless they do unit testing and paired programming, or something similarly effective at clearing out the knucklehead bugs before it comes to you, then you are going to be overwhelmed with bugs to find, and that will slow you down. You'll be like one of those contestants in a "money booth" trying to grab flying cash while it whirlwinds all over the place. If you like sports metaphors, tell them you want be a good goalie for the team, but even you can't defend against multiple shots on goal all at the same time-- or when your own team is trying to score goals on you.
- **Avoid bureaucratic bug reporting protocols.** Bug investigation and reporting take up a lot of your time. The more time it takes, the less time you have to find new bugs. But if you test alongside the programmer whose code it is, you'll find that reproducing the bug and writing a report become mostly unnecessary. He sees what you're doing and immediately reacts to it. I've found this to be a wonderful way to test something in its early stages. I've also experienced a project where my testers were required by our client to use a cumbersome bug tracking system that reminded us of filing tax forms. We estimated that we lost 30 minutes of test time for each bug we reported, over and above

the time it took us to actually investigate and describe the steps to reproduce. On top of that, we were not allowed to talk to the programmers. We didn't work near them, never met them, and heard from them. It was an outrageous waste of time. If you are a team of one, tell your management that you need to minimize your report grinding and maximize your bug finding.

My Favorite Advice for Omega Testers...

If you are an omega tester, feeling surrounded by creatures who have tester-like qualities (arms, legs, human DNA, etc.) and yet are not like you, do not be dismayed. Befriend them. Get them to help you. Here's the secret: people will help you test if you make it easy and fun for them. Let's call these people beta testers.

Beta testers can help you test, but *don't give them any paperwork to do*. Blech. I was once a beta tester on an early PDA (one of the very earliest ever, which was called the Magic Link or something like that). I noticed only after having the device for a couple of weeks that Sony had expected me to fill out a minute-by-minute log of exactly what I did with their product. Ha ha ha. No, I just doodled in some fake data and returned it.

Many things are expected of a true professional tester. You should be able to report and justify your test coverage and your oracles. You need to understand risks and use test tools. But push none of that onto your beta testers. Instead, create a carnival atmosphere of old fashioned bug huntin'. If I can get my tester helpers all in the same room for a testing event, I make sure they are fed, I pair them up, and give each pair a bell to ring when they think they found a problem. Beta testers are motivated by the respect they receive and the feeling of being listened to.

I've also had very good experiences with bug bounties-- paying people who are not on the test team for each critical bug that they find. You can do this with actual cash money, or do it with play money that can be used to purchase items at an auction you hold at the end of the project (the latter method allows you to completely control the budget).

Who's the *Real* Zombie?

If you do a good job being an omega tester, you'll find that the other people on the team, over time (it may take several projects), will come to respect and rely upon you. Otherwise, especially if you fall into a formulaic, uninspired, or complainy approach to your work, the project team may come to see *you* as the mutant zombie in their midst. One sign of this is that they run away when you shuffle toward them with your arms stiffly out as if groping for brain matter or "unambiguous requirements." Don't be like that, okay?