Is This Testable? A Personal Journey to Learn How to Ask Better Questions from My Applications and Engineering Team

Michael Larsen

mkltesthead@gmail.com

Abstract

Have you ever felt like you were working at cross purposes with an application you were developing and testing? Have you wished that there were easier ways to interact with your application? Do you struggle with the fact that efforts to automate your application under development yield less than satisfactory results? Perhaps you discover that for every answer or solution what results is more and more questions. I decided to undergo a process of better understanding of what made the applications I work with testable in the first place. Through that process, I've learned a few things that may be helpful to anyone looking to make their product more responsive to both human and external program interactions.

Biography

Michael Larsen is a Senior Quality Assurance Engineer with Socialtext/PeopleFluent. Over the past two decades, he has been involved in software testing for a range of products and industries, including network routers & switches, virtual machines, capacitance touch devices, video games, and client/server, distributed database & web applications.

Michael is a Black Belt in the Miagi-Do School of Software Testing, helped start and facilitate the Americas chapter of Weekend Testing, is a former Chair of the Education Special Interest Group with the Association for Software Testing (AST), a lead instructor of the Black Box Software Testing courses through AST, and former Board Member and President of AST. Michael writes the TESTHEAD blog (http://mkltesthead.com) and can be found on Twitter at @mkltesthead. A list of books, articles, papers, and presentations can be seen at http://www.linkedin.com/in/mkltesthead.

1. Introduction

Testability is a topic that can take many pages and many experiences to cover. My experiences with Testability came into focus by working through the “Thirty Days of Testability” challenge that the Ministry of Testing put together. By going through this process, I learned a great deal about what makes applications and services testable. Testability is not some unimaginable outcome. It requires communication, involvement, and a willingness to look at a product objectively and determine what would help make it more testable and, potentially, more usable by everyone.

2 Making Testability a Primary Focus

Testability can be described as:

> ”The logical property that is variously described as contingency, defeasibility, or falsifiability, which means that counterexamples to the hypothesis are logically possible. The practical feasibility of observing a reproducible series of such counterexamples if they do exist.”

> “In short, a hypothesis is testable if there is some real possibility of deciding whether it is true or false based on real experience. Upon this property of its constituent hypotheses rests the ability to decide whether a theory can be supported or falsified by the data of actual experience. If hypotheses are tested, initial results may also be labeled inconclusive.”

That may seem like overkill when it comes to testing software. In truth, I consider it a great place to start. What is the goal of any test that we want to perform? We want to determine if something can be proven correct or refuted. Thus, we need to create conditions where a hypothesis can be either proven or refuted.

How we do that will vary from project to project. Simple programs may be considered with a few examples and paths through the visible interface. More complex applications may require interacting with a variety of interfaces, including but not limited to:

- operating systems
- desktop and mobile platforms
- desktop and mobile applications
- web browsers
- web servers
- database servers
- load balancers
- third-party applications and libraries
- cloud devices

Regardless of the level of complexity of application, the goal is the same. We either confirm our hypothesis is correct or we prove our hypothesis is wrong. If we cannot verify our hypothesis, then we must call into question the very methods which we will examine why it isn't going to work for us. What do we do if we determine we can’t? For me, the question of “testability” falls into this area, addressing why

we cannot effectively confirm or refute a hypothesis. In short, if there is an aspect that acts as a barrier or lessens our effectiveness to be able to perform actual experiments and make decisions based on the data provided, we have testability issues we need to examine.

3. Getting Specific with Hypotheses Using Data

Andrew Peterson 8/20/2019 4:32 PM
Comment [1]:

Andrew Peterson 8/20/2019 4:32 PM
Comment [2]: Extra line space after the title I think.

One of the aspects that I think is important is to look at ways that we can determine if something can be performed or verified. At times that may simply be our interactions and our observations. Let's take something like the color contrast on a page. I can subjectively say that a light gray text over dark gray background doesn't provide a significant amount of color contrast. Is that hypothesis reasonable? Sure. I can look at it and say, "it doesn't have a high enough contrast."

That is a subjective declaration based on observation and opinion. Is it testable? As I've stated it, no, not really. What I have done is made a personal observation and declared an opinion. It may sway other opinions but it's not really a test in the classic sense.

What's missing? Data. What kind of data? A way to determine the actual contrast level of the background versus the text. Can we do that? Yes, we can, if we are using a web page example and we have a way to reference the values of the specific colors.

Since colors can be represented by hexadecimal values or RGB numerical values, we can make an objective observation as to the differences in various colors. By comparing the values of the dark gray background and the light gray text, we can determine what level of contrast exists between the two colors. Whether we are using a program that can create a comparison or an application that can print out a color contrast comparison, what we have is an objective value that we can share and compare with others.

"These two colors look to be too close together"... this is not a testable hypothesis.

"These two colors have a contrast ratio of 2.5:1 and we are looking for a contrast ratio of 4.5:1 at the minimum" ... this is a testable hypothesis and it also contains data points to support it.

Andrew Peterson 8/20/2019 4:33 PM
Comment [3]: Too much spacing before and after this Title

4. What Do Your Logs Say?

Some of the most effective and often overlooked sources of data for how an application performs are the application log files. One of the reasons that log files are often overlooked is the sheer amount of information present. Log files are great places to see the history of events and actions that have taken place in a system. The downside is that the sheer volume of information can be overwhelming.

Additionally, it is important to have a clear understanding of what I mean by “the application” when I discuss log files. Do I mean the actual application I work with? How about the extended suite of applications that plug into ours? I mention this because, for each of the components that make up our application, there is a log file or, in some instances, several log files to examine.

To use one of the products I have worked on, we have a large log file that is meant to cover most of our interactions. Even then, there are so many things that fly past that it can be a challenge to figure out exactly what is being represented. Additionally, there are logs for a number of aspects of our application and they are kept in separate files, such as:

- installation and upgrades

- authentication
- component operations
- third-party plug-ins
- mail daemons
- web server logs
- search engine logs

and so on.

There are a variety of ways to parse this information and examine it with scripts after the fact but that requires that we understand what we are looking for in the first place. As for me, looking in log files is a voyage of discovery.

I have found that using simple screen multiplexers such as “screen”, “tmux” or “byobu” and splitting one of my windows into multiple fragments allows me a clear look at a variety of log files at the same time. This way, I can see what is actually happening at any given time from a variety of perspective. Additionally, I get used to seeing which log files post updates after system interactions.

Some logs fly by so fast that I have to look at individual timestamps to see dozens of entries corresponding to a single second, while other logs get updated very infrequently, usually only when an error has occurred.

Putting together an aggregator function so that I can query all of the files at the same time and look for what is happening can be a plus but only if they use a similar format. Unfortunately, this is a common problem. If we have multiple log files, it would be helpful to have all of the resulting log files be formatted in a similar way, such as the following example:

log_name: timestamp: alert_level: module: message

repeated for each log file.

Let’s use the example of a user updating a database field. It’s possible that this action might be reflected in multiple places. It might show up in a log for the web server, for the database server, for the backup server and perhaps other locations where the log will be recorded (search results, access logs, error logs, etc.).

Having an option to gather all log files into an archive and have them archived each day (or whatever time option makes the most sense) also provides considerable benefits. If at all possible, try to make the messages put into the log files human-readable.

5. Identify a “Partner in Crime”

If I had to find out what the three biggest customer impacting issues related to my product are, how would I get that information? Working with support engineers and sales engineers should be a first line of contact for every software tester or person involved with software quality.

I am fortunate in that I have a terrific point of contact in the guise of a great customer engagement

engineer. I can count on one hand the number of times when I have announced an update on our staging server and not heard a reply from this person of, "hey, what's new on staging?" They make it their responsibility to be clear and up to date with every single feature and every option available in the product and when it is deployed. They also make a lot of sample accounts and customizations to our product to push the edges of what the product can actually do.

One of the examples we use frequently is a site that is cloned from one of our most active customers. We set up our test environment to match theirs in every aspect. These include the number of front-end servers, load balancing devices, back end servers, database replication schemes, customer accounts, and views to data and applications that we display through widgets in the application dashboard.

By putting together these environments and creating examples that reflect what our customers actually see, we can more effectively look at the methods and means to interact with those systems the way our customers do, not necessarily in the ways we think or would like them to.

6. Are We Using Feature Analytics?

In my company, we have two applications we use and monitor for Analytics. One is for overall HTTP traffic and demographics:

- what pages get hit the most?
- what browsers are used the most?
- what time of days do people most use the application?
- how many parallel connections are we maintaining at any given time?

Andrew Peterson 8/21/2019 4:25 PM
Comment [4]: Should this be Ariel 10?

This helps me define what I should prioritize in my testing, as well as to the load our application might be subjected to. By addressing or considering these areas, we can ask specific questions or develop experiments that will help to provide answers. How many concurrent users will be accessing the system? How will we know? What processes will be running during those interactions? Do we have enough physical or virtual memory to interact with those processes?

Andrew Peterson 8/21/2019 4:26 PM
Comment [5]: comma

The second tool we use is based on feature analytics, as in "what features in our product are our customers actually using?" As an example, there have been situations where customers have asked for specific features to be included in our application. These requests range from “this would be nice to have” all the way to “if this feature is not included, we cannot purchase the product.” What is a development team to do? Depending on the level of feedback and urgency, it is possible that a great deal of effort will be put into developing the features that have been requested. There are times where this makes a lot of sense, such as when a feature (take Responsive Design as an example) may be the make or break feature for a large organization. The want to have a site that will allow for display on multiple devices and not have to rely on separate apps for desktop and mobile. With tens of thousands of users, making a feature development priority is easy to in this case.

Andrew Peterson 8/21/2019 4:27 PM
Comment [6]: Should this be grouped in the same paragraph?

At other times, we have had requests that have been labeled as very important to make a sale for a large organization, only to later see that the adoption of that feature was low to non-existent. Additionally, applications or pieces of functionality that may think might be the most significant may be to a small but vocal subset. How do we actually determine if these features are actually being used? By having feature level analytics that can be examined and monitored, it is possible to both determine what features are actually being used at a variety of customer sites as well as see trends of what are not being used. This can help us determine the priority of testing efforts and areas that both need to be tested as well as make decisions about future development efforts or even support for these components.

Looking back to the original definition of proving a hypothesis, sometimes the most important hypothesis to prove is “does it make sense to keep a piece of functionality working the way that it is?” We can make

guesses but implementing feature analytics can give us data that we can them make actionable decisions about. If features are actively being used, we know that efforts for continued feature enhancement is desired and necessary. IF features are not, we can objectively evaluate how and when they are being used, or how little, and make a determination as to whether or not they should be gracefully deprecated or removed from the product completely.

7. Automateable Does Not Necessarily Mean Testable

Alan Richardson has written an effective blog post around the differences between “Automateable and Testable.” (Richardson, 2019). Just because you can automate something doesn't necessarily means it's testable. Automat-ability means that something else (a program, a shell script, or an API) can interact with it. However, while that is capable to be done, it doesn't necessarily mean that it's going to be a good experience for a user. It also does not mean that the hypothesis related to the test will necessarily be proven or refuted.

"Testability is for humans. Automatability (Automatizability) is for applications.” (Richardson, 2019)

An example can be seen with an applications menu bar. It is possible to click on the various elements and open other pages within the application. Those interactions can be automated. However, those interactions give us no indication if that path makes the most sense or if the design of that interaction could be improved. Can we select each of the elements in the navigation bar and then move to another page? The answer is yes if we click with a mouse or call out the elements and click them with an automated script. Can we navigate between the pages using nothing but a keyboard? Based on traditional automation techniques, we may or may not know if that is true.

Testability can add a number of options to a program and can help to make a system more enjoyable for a user. It may help with making an application more automateable but it doesn't guarantee that it will.

8. Ask the following question: “How Am I Going to Test This?”

In my organization, stories get defined and features are determined during our story kickoff or design workshop. As a quality advocates, it is my duty to ask, “how am I going to test this?” every time I am part of these meetings. What might be the results of some of these questions? It’s possible that I might determine it would be helpful to:

- add calls to the API so that I can query and set values without having to fire up a browser and look at a bunch of things on the screen.
- create actual configuration enhancements of an application so that I can tweak various things while I'm setting up a test environment.
- make sure that elements are named well (whenever possible) and that duplication is minimal.
- ensure that the error output is understandable and helps me understand what has happened and what I can do about the error.

Testability hooks are often put in at points where testing might prove difficult, but we won’t be able to determine where those difficult areas are if we don’t have these conversations. Also, there may be friction between the development team and the testing team if these conversations are not held early in the development process. It is much easier to add a testing hook early in the process rather than later or as

an afterthought. To paraphrase a Chinese proverb, “the best time to plant a shade tree was twenty years ago. The second-best time to do it is now.”

When I have had discussions about testability issues, I have found the greatest successes come from examining areas that have led to bugs. While it is not productive to say, “why didn’t you find that bug?”, it is often helpful to examine where we might have prevented it or how we might prevent it going forward. Without laying blame, I have found that these are opportunities to look at ways of introducing testability hooks, since rework is already underway for that particular issue.

9. Get a Handle on Your Test Data

Depending on the product under test, the data we need to effectively test that application may be simple or complex. In my day-to-day testing work I deal with a lot of users and data that would be generated by those users.

At the basic level, everything is associated with an account, so often the most effective method to load test data (as well as to protect it and to use it from a known starting point) is to import an account that is already set up with the information I want to use. I actually have several of these and each is set up to help me deal with a variety of issues. I have accounts created for Localization, Responsive Design, Accessibility & Inclusive Design, and Large Customer simulation.

Additionally, I have data that deals with the components of our system so that I don't have to constantly reconfigure those elements (that includes text examples, HTML and Markup formatting, videos, user details, language preferences, etc.). The key here is that I try to limit the use of test data that tries to be all things to all circumstances. While it can be helpful to include a lot of details in one place, it can also complicate the situation in that there is "too much of a good thing” as well as too many variables to chase.

Another way that I try to keep test data useful and fresh is that I determine the methods that can best generate the data that I use and help me to keep track of everything in a noticeable way. One of those methods is that I have general and specific scenarios. When I do tests with large numbers of users I generate that data with a tool called "Fake Name Generator". This has been my go-to tool for more than a decade and it provides individual details I can call up one at a time to use, or I can get bulk downloads with tens, hundreds, or thousands of users. The system limits you to 100,000 users for any given request, but over time, it is possible to generate several hundred thousand or even millions of users.

10. Conduct Regular “Show and Tell” Sessions

As the release manager and build manager for each of our releases, one of my responsibilities has been to gather a cross-section of the broader engineering, sales, and support teams so that we can show them the new stuff going out with our next release.

Additionally, we have as part of our "definition of done" to demo any feature we are working on to our product owner and/or our chief sales engineer. This is a great time to walk through the feature, show as many parameters as we can and to listen to their questions. Much of the time it's a fairly straightforward "show and tell" but every once in a while, we have some deeper discussions. These longer conversations often serve to point out that the new feature we are demonstrating may have some follow-on stories to consider. Often, it's a chance to see and determine how well we have answered the product owner’s/representative’s expectations so that we can adjust accordingly in the future. It is also a good opportunity to make sure that the programming and testing efforts are focused on the same goal or are at least in the same ballpark.

Andrew Peterson 7/25/2019 5:06 PM
Comment [7]:

Larsen, Michael 8/19/2019 11:15 PM
Comment [8]:

Larsen, Michael 8/19/2019 11:15 PM
Comment [9]:

These sessions frequently help us to ask the earlier considered question of “How am I going to test this?”

11. Pair with Your Developers and Test Together

I'm fortunate in that I have developers that are willing to pair and look at stuff with me. As we have G-Suite as our main communication platform among the team, it's easy to demo stuff and talk out what is being tested. We are a small team and technically speaking, with management out of the equation, our engineering team is equally balanced between programmers and testers. Thus, it's not an imposition or overly burdensome to get together and pair with developers. Another opportunity that is available is to look at the code and examine unit tests that have been created or need to be created. By doing so, we can have a starting point to see what the programmers have determined is important to be tested at this level and to help initiate conversations about what else could or should be covered or to identify areas where there may be gaps.

Andrew Peterson 7/25/2019 5:15 PM
Comment [10]:

Comment [11]:

Lisa Crispin has written a handy guide on "Pairing with Developers: A Guide for Testers" (Ministry of Testing, 2019). Lisa’s article is a great resource and while I have little to add to it specifically, I can share a few things that I have found help considerably with pairing with developers.

Andrew Peterson 7/25/2019 5:07 PM
Comment [12]: Ministry

1. Have a specific goal for the session: By developing a very specific charter around an area that is either difficult to test or that may require more knowledge that the developer has, being super specific really helps with this process. Often when I approach and have a question that has very clear parameters (or as clear as I can make them) I'm much more likely to be able to block out time with them. More times than not, we range farther than that specific area because we're into it and we're both able to get more done together than separately.

2. Block out a specific time interval: set an appointment, set the desired time (about an hour is my usual suggestion, no more than two). We frequently blow past the single hour time set aside but we're usually good about heeding the two-hour limit. Beyond two hours we tend to lose focus but up to that point is usually very effective.

3. Do your homework in advance: This goes with number 1, but if I'm going to try to understand something that's going on in the code, it helps for me to have reviewed it first (if possible). Greenfield development efforts don't always allow for that but since our current efforts are mostly focused on revamping existing functionality, there's plenty of opportunities for me to read up and understand the parameters the developers are working with. I may not understand all of it but at least I am ready to start sessions with a list of questions.

12. Identify Your Dependencies

Most modern applications are not self-contained. They rely on a variety of third-party or open source components to allow them to run, as well as external libraries and other dependencies that may not be clearly outlined. If an application uses a micro-services architecture, it adds to the complexity of testing when there are external groups that control what is being displayed and how it looks or if it's even available. This is the case with my company currently and we are actively trying to deal with it.

To this end, one of the more helpful approaches I have found is to create a specific test environment with users that have permissions to access multiple connecting applications. This allows for the micro services applications to interact with each other in a meaningful way and to allow me to examine more of the interactions between systems. This way I can determine how many issues are specifically associated with our product and which issues are associated with other applications my product is interacting with.

13. Check Your Source Control History and "Feel the Heat"

A term I discuss with my team frequently is to "look for the heat" in the source code. That is to say, where are we making frequent changes? What other components do those changes interact with? The closer the level of interaction, the greater the level of "heat". The farther away, the less heat there is, or likelihood that those components will be involved in the most recent changes. In short, where there is heat is where we need to focus our testing efforts. By doing that, by examining existing unit tests, integration tests and end to end tests (if they exist) we can see if we are effectively covering these areas or if more specific testing is needed. Again, going back to the original definition, do we have enough information? Do we have enough of a handle on the capabilities and inner workings on these areas to determine if the changes made are working as we want them to?

Andrew Peterson 7/25/2019 5:19 PM
Comment [13]: h

By using our source control history, it is possible to find out which parts of our system change most often.

As a recent example, we have been involved in an update to a more responsive design. To that end, it means that there is a lot of change to our front end and how it is displayed with regard to the device or user agent making the calls. By contrast, our legacy interface changes very little and the workflows, IDs and respective interactions are well defined and rarely change. However, there are certainly instances where new code does interact with components that effect that older code and thus have an effect on how that legacy interface responds. By looking at the source code changes and getting a feel for what has been modified, I can also determine which areas in our legacy interface might need to be looked at once again.

14. Conclusion

There are a variety of methods and tools that we as quality engineers and quality advocates can leverage to help make our products more testable. It all starts with us being able to identify areas that are potential issues, as well as being able to talk about them in a meaningful way with our development teams. Additionally, we need to be willing and able to look at our systems objectively and identify the areas we can test effectively first, so that we can better define and describe the areas that have deficiencies. There is no easy fix to testability. The process is ongoing. It requires focus and consistent communication. It requires a willingness and an ability to encourage the development team to likewise consider testing the product actively and looking at it from a variety of perspectives.

While the process may be slow and challenging, with consistent effort and a willingness to keep asking questions, it will be possible to narrow down the areas where we have to as "is this testable?" It may not be possible to make every area perfectly testable, but we can certainly make our applications more testable over time and that is an outcome I think everyone would welcome.

References

Ministry of Testing. “Thirty Days of Testability”. Ministry of Testing, https://www.ministryoftesting.com/dojo/lessons/30-days-of-testability (accessed July 20, 2019).

Wikipedia. “Software Testability.” https://en.wikipedia.org/wiki/Software_testability (accessed July 20, 2019).

Alan Richardson. “Testability vs Automatability - in theory”. Evil Tester, https://www.eviltester.com/2018/01/testability-vs-automatability-in-theory.html (accessed July 20, 2019).

Fake Name Generator. https://www.fakenamegenerator.com/ (accessed July 20, 2019).

Crispin, Lisa. “Pairing With Developers: A Guide For Testers”. Ministry of Testing. https://www.ministryoftesting.com/dojo/lessons/pairing-with-developers-a-guide-for-testers (accessed July 20, 2019).