CK-12

flexbook
next generation textbooks

CK-12 FlexBook

What is Testing?

To access a customizable version of this book, as well as other interactive content, visit www.ck12.org

CK-12 Foundation is a non-profit organization with a mission to reduce the cost of textbook materials for the K-12 market both in the U.S. and worldwide. Using an open-content, web-based collaborative model termed the **FlexBook®**, CK-12 intends to pioneer the generation and distribution of high-quality educational content that will serve both as core text as well as provide an adaptive environment for learning, powered through the **FlexBook Platform®**.

Printed: April 1, 2013

flexbook
next generation textbooks

CONTRIBUTORS
summerqamp.org
CK12.org

Contents

CHAPTER 1

Testing is Exploration: TEST THIS!

This module is meant to give testers a chance to test something with the skills they already have, no matter their starting point. This exercise is meant to help testers understand how exploration helps them in their goals of understanding how a product works and what might be issues to consider.

Before we talk any theory, before we discuss any buzzwords or jargon, before we do any of that, we want to give you an experience of looking at an application and figuring out what it does.

TEST THIS: Black Box Machines

James Lyndsay has a number of "Black Box Machines" that he has made available for testing simulations.

They can be reached here:

http://www.workroom-productions.com/black_box_machines.html

Your Mission: Using any abilities you have and anything you can find on the page or in the Flash applications themselves:

1. Determine what the machine does.

2. See if you can find any issues with how the machines "do their job".

Take time to explore each of the machines, and take notes on what you find. When you are finished, continue on to the next section.

CHAPTER 2 What is Software Testing?

What is Software Testing?

Software testing is, in general, the actions performed to gather information about a software application, which is then used to determine if it is fit for public or widespread use.

It is, in the simplest terms, a tester asking a product questions, and in turn getting answers. It is based on these answers that the tester asks further questions, and gets further answers.

That's it. That's the whole of what software testing is, but just saying that does not come even close to what the software tester *actually does or needs to know to be effective*.

The ways that we ask those questions, the techniques, are broad and varied.

Software testers use a variety of words to describe what it is they do. These words are:

Verification: we verify that a product is working the way that it should ("should" means that there is some way to actually verify that the product is behaving the way that it was designed).

Spotting Errors: we look to see if we can "trip up" the application. This is often referred to as "see if you can break it", but that's only part of the focus. There is also seeing how the application deals with errors. If we do something deliberately wrong, will the program tell us that? If so, will it tell us in ways that are meaningful?

Delivering Value: We can verify that the system is doing what it was designed to do, and we can see the way the program deals with how we try to break it. Outside of that, an even more important area to consider is "does this really meet the need of our customers?" If we make a product that does what the design documents say it should do, and if we see that the program responds well if we try to "behave badly" while testing it, does any of that matter if our customers ultimately hate using the end product?

CHAPTER 3 Testers Ask Questions

Understanding the difference between an unhelpful and vague question, and asking questions that are targeted and specific.

How to Ask Questions the Right Way

I think it's impossible to ask "what is Software Testing?" without also focusing on "how to ask the right questions". It's entirely possible to push buttons and "ham fist" our way to problems, but are those problems significant? Do they really matter to the end user or the stakeholders? They may, they may not. The problem is that, if we report a lot of "unimportant bugs", or if we ask "insubstantial questions" regularly, our knowledge and efforts will often be dismissed, or at the least given lower priority.

Why?

Because we are asking the wrong questions. More accurately, we are asking questions in an ineffective way.

The recommendations listed below comefrom a sitethat isdedicated to helping answerquestions on various hacker forums.

Note: **Hacker** in this case means anyone that works with code and tries to make changes, modifications and enhancements to see what those changes do. This is not to be confused with a "Cracker" who works to disrupt services, steal information or break into systems for fun or dishonest profit. From the site:

"So, while it isn't necessary to already be technically competent to get attention from us, it is necessary to demonstrate the kind of attitude that leads to competence — alert, thoughtful, observant, willing to be an active partner in developing a solution."

Testers need to approach problems or issues in the same way.

Before You Ask

Let's approach this from the aspect of "before you start testing". What are some things you might want to make sure that you know?

You may want to make sure you are aware of industry standards or what other organizations who make similar software are doing.

You may want to understand alternative ways of doing things that makes sense in that particular world. Testing a photo editor will be much more focused and understandable if you understand actual photographic techniques. In testing, we refer to this as having "domain knowledge", or at least an understanding of where the area we are going to test exist.

How can we be sure we are asking the right questions?

1. Try to find an answer by searching (internal) the existing documents, specs, issue tracking system, or (external) forums that deal with this kind of application(s).

2. Try to find an answer by searching the Web.

3. Try to find an answer by reading the product manual (if there is one).

4. Try to find an answer by reading a FAQ (if one exists).

5. Try to find an answer by inspection or experimentation.

6. Try to find an answer by asking a skilled friend.

7. Try to find an answer by reading the source code (if you have that skill and have access to it).

When you ask your question (whether it be a product or a person), display the fact that you have done these things first. Better yet, display what you have learned from doing these things. Include things like "I searched google and could not find an answer based on [words used]. . . " This helps set the parameters that others will understand.

Take your time.

Do not expect to be able to solve a complicated problem with a few seconds of searching.

Read and understand. Give the problem some thought before approaching experts (or before testing or reporting bugs). Trust us, they will be able to tell from your questions how much reading and thinking you did, and will be more willing to help if you come prepared.

Prepare your question. Think it through. Hasty-sounding questions get hasty answers or none at all.

Beware of asking the wrong question or one based on faulty assumptions. You will earn an answer (from a product under test or from an expert), if you earn it, by asking a substantial, interesting, and thought-provoking question.

When asking individuals for clarification, make it clear you are able and willing to help in the process of developing the solution.

“Would someone provide a pointer?”

“What is my example missing?”

“What should I have checked?”

Are more likely to get answered than

“Please post the exact procedure I should use.”

By doing the former, you are making it clear that you’re truly willing to complete the process if someone can just point you in the right direction.

Examples of Good/Bad Questions (from previous site)

Good and Bad Questions

The following text is almost verbatim from the above referenced site. Let’s discuss if we want to change them or present them as is. I think with proper attribution, we can present them as is, as I think the tone of the questions and responses are good.

These are questions about the same problem, one asked in a stupid way and one in a smart way.

Stupid: Where can I find out stuff about the Foonly Flurbamatic?

This question just begs for a response of "RTFM" or "STFW" as a reply. These stand for "Read the [Expletive Deleted] Manual" or "Search the [Expletive Deleted] Web". For this course, I’ll instead use the much kinder, albeit still slightly sarcastic acronym "LMGT". This stands for "Let Me Google That", which gets the same message across. Developers help those who help themselves, and you will get a variation on these types of responses if you ask a question based on something you could have easily looked up yourself.

Smart: I used Google to try to find “Foonly Flurbamatic 2600” on the Web, but I got no useful hits. Can I get a pointer to programming information on this device?

This one has already done the LMGT step, and has come up empty handed. This simple step alone will change the responses the tester gets, because they have invested some of their time and energy to learn about the product.

Stupid: I can’t get the code from project foo to compile. Why is it broken?

This question is placing the blame on someone else. While that may be the case, it’s bad form to automatically assume that someone else has done something wrong. It also shows a lack of humility, and the developers in question will respond in kind.

Smart: The code from project foo doesn't compile under Nulix version 6.2. I've read the FAQ, but it doesn't have anything in it about Nulix-related problems. Here's a transcript of my compilation attempt; is it something I did?

Notice the difference in tone. The first thing to see is that they have specified the steps they have already taken, their experience with looking at the FAQ, and most importantly, not assuming that the problem is external to them (i.e. they are not blaming other people, but asking if they have missed something).

Stupid: I'm having problems with my motherboard. Can anybody help?

The first thing to notice about this question is. . . what? More like a lot of whats. What type of motherboard? How is it installed? What CPU or RAM are you using? What CMOS? And a bunch of other questions that we would have to ask just to get to the point of being able to answer even a small part of the possibilities. Most won't even bother, and will likely just ignore the question.

Smart: I tried X, Y, and Z on the S2464 motherboard. When that didn't work, I tried A, B, and C. Note the curious symptom when I tried C. Obviously the florbish is grommicking, but the results aren't what one might expect. What are the usual causes of grommicking on Athlon MP motherboards? Anybody got ideas for more tests I can run to pin down the problem?

This is good overall protocol whenever dealing with an issue of any kind. In fact, the above question would make for a good bug entry all by itself (more on bug reporting later).

The reason this set of questions will likely get attention is that the tester has applied a problem-solving approach to the issues they are seeing. They have reported what they tried to do to make it work, and has also made a guess as to what the problem might be. They may be wrong, but it shows the programmer that they are working towards isolating the issues and understanding the problem.

Additionally, instead of saying "Give me an answer", the tester is asking "Please help me figure out what additional diagnostics I can run to understand the problem"

Excerpted from:

How To Ask Questions The Smart Way

http://catb.org/esr/faqs/smart-questions.html

CHAPTER 4

What Do Testers Find?

What do Tester's Find?

This section comes from two blog posts written by James Bach and Michael Bolton, respectively.

Some of you may be familiar with the 2003 movie "Holes". If not, I won't spoil the plot for you, but the main point is that a number of youth are serving at a juvenile detention camp. While there, they are made each day to dig a hole that's five feet round, primarily to "build character", but to also let the warden know if they "find anything interesting".

Yes, sometimes, software testing can feel like this. We do a lot of digging, and sometimes it is repetitious, but quite often, we are rewarded for our digging when we "find something interesting".

So what constitutes "interesting" to a tester? When we are looking for something in a software application, what are we likely to find?

Testers find bugs.

We look for anything that threatens the value of the product. "Quality means value to some person, at some time, which matters". –Jerry Weinberg

From James Blog: Some people like to say that testers find "defects." That is also true, but I avoid that word. It tends to make programmers and lawyers upset, and I have trouble enough.

Example: a list of countries in a form is missing "France."

Testers find risks.

We notice situations that seem likely to produce bugs. We notice behaviors of the product that look likely to go wrong in important ways, even if we haven't yet seen that happen.

Example: A web form is using a deprecated HTML tag, which works fine in current browsers, but may stop working in future browsers. This suggests that we ought to do a validation scan. Maybe there are more things like that on the site.

Testers find issues.

An issue is something that threatens the value of the project, rather than the product itself.

Example: There's a lot of real-time content on eBay. Ads and such. Am I supposed to test that stuff? How should I test it?

Testers find testability problems.

It's a kind of issue, but it's worth highlighting. Testers should point out aspects of the product that make it hard to observe and hard to control. There may be small things that the developers can do (adding scriptable interfaces and log files, for instance) that can improve testability. If you don't ask for testability, it's your fault that you don't get it.

Example: You're staring at a readout that changes five times a second, wondering how to tell if it's presenting accurate figures. For that, you need a log file.

Testers find artifacts.

Also a kind of issue, but also worth highlighting, we sometimes see things that look like problems, but turn out to be manifestations of how we happen to be testing.

Example: I'm getting certificate errors on the site, but it turns out to be an interaction between the site and Burp Proxy, which is my recording tool.

Testers find curios.

We notice surprising and interesting things about our products that don't threaten the value of the product, but may suggest hidden features or interesting ways of using the product. Some of them may represent features that the programmers themselves don't know about. They may also suggest new ways of testing.

Example: Hmm. I notice that a lot of complex content is stored in Iframes on eBay. Maybe I can do a scan for Iframes and systematically discover important scripts that I need to test.

[Editors Note: These examples are placeholders and are the exact entries from the two blog posts. This list needs to be reviewed and potentially different examples used.]

Credits:

WHAT TESTERS FIND (James Bach)

http://www.satisfice.com/blog/archives/572

MORE OF WHAT TESTERS FIND (Michael Bolton)

http://www.developsense.com/blog/2011/04/more-of-what-testers-find-part-ii/

CHAPTER 5 Understanding Programming: TEST THIS!

Talking about software testing without understanding how software is made can lead to many misconceptions. The act of making software and writing the code that computers can read and execute is called "programming".

The challenge is that, without deciding on a language that everyone will use, it can be difficult to understand programming concepts and how they are applied.

Instead, we want to have you play a game.

TEST THIS: Lightbot

Lightbot is a game that allows the player to "program" the movements of a robot to walk, turn, jump, and maneuver through obstacles so that they can meet objectives (usually "lighting up" certain tiles, hence the name).
http://armorgames.com/play/6061/light-bot-20
Your Mission:
Play through the game, completing as many of the levels as possible. While you do so, take note of the numerous programming ideas that are demonstrated, such as conditions, functions, recursion, etc.). Be prepared to explain how these steps resemble computer programming, and how many of the ideas apply to the way that software is made today.

So why did we have you play this game? What did you notice in the process? Were you able to advance in the game? How far? Did you notice that many of the commands couldn't work unless you made small libraries (functions) to hold a bunch of the commands?

For those with a lot of programming experience, this may seem very basic, but for many people who have not programmed, this is a good way to help explain and practice the concepts that frame all programming languages.

[Next Section: A Primer on Programming, Super Basic. How much should we put in?]

CHAPTER 6

What Does Quality Assurance Mean?

Understanding the objectives of Quality Assurance, and dealing with the misconceptions as to what Quality Assurance actually means.

What does Quality Assurance Mean?

This is a loaded phrase, and it means many different things to different groups of people.

The most valuable piece of information any software tester can learn is that they themselves cannot Assure Quality.

Stop.

Go back and reread that last sentence.

Have you done that?

Great. Now go back and read it again.

Quality Assurance is not a department. It is a process an entire company has to practice to actually happen. Software Testers cannot "assure quality". What we can do is:

- **provide information about quality**
- **inform and alert key individuals that we have found something that concerns us.**
- **if we are sure of the problem, recommend that it be fixed.**

Notice the words used there: provide information, inform, alert, investigate, report, recommend

What words are not used?

Words like block, prevent, rewrite, fix, deploy.

Why aren't they used? Because we who test software rarely, if ever, have the ability to do these things.

Programmers can do these things, as can systems administrators and deployment specialists. Software testers, unless they are the programmers for a particular area, usually do not have the ability to make direct changes. If they cannot make changes, can they rightly say they are able to assure quality? NO!

We can suggest that an issue is significant enough to stop production, but we as testers do not have the final authority as to whether or not a feature gets fixed or doesn't.

Having said all that, Quality Assurance is a term that is very entrenched in the thoughts and the vocabulary of testing. It's a short-hand for the things that testers do to help provide information and let programmers and management see and consider what they will do.

We could say that "Q.A." really stands for "Quality Assistance" or "Question and Answer". Those are things that we can control and that we can contribute to the discussion.

CHAPTER 7 Attributes of Good Testers

What makes a good Software Tester? Phrases like "attention to detail", "ability to work with complex systems" and "enjoy solving problems" are often bandied about, but they only tell a small part of the story. There's a lot of aspects that come into play when we consider what makes a good tester, and many people get into software testing for a variety of reasons.

It is rare to find someone who has deliberately gone into software testing as a first choice of career. What I mean by that is that, it is uncommon to hear of someone who goes to school, makes a plan to become a software tester, and then goes out and devotes their education and efforts to specifically getting a job as a software tester in a company. For most of us, if you were to ask "How did you become a software tester?", the most common answer you would hear is "I just sort of fell into it."

Truth be told, "I just fell into it" is also rarely true, but that is how it often feels when we look back in hindsight. More times than not, what happens is that there is an opportunity that we recognize that needs to be filled, we discover that we are effective in the testing role, and based on that feedback, we continue testing, and grow into the role over time.

Many people look at software testing as a stepping stone to doing other things. For some, it may be a starting point towards becoming a programmer. For others, they may want to become product managers. Others may want to go into product marketing. Others may want to become Customer Support representatives (and to be fair, a lot of really good software testers also come from Customer Support roles).

Those who choose to stay with software testing and make it a career often have a number of things in common:

- They are often top down, big picture thinkers who tend to approach problems starting from the whole and diving in to understand the smaller parts.
- They like to ask questions, and see how those questions are answered.
- They have a high level of comfort with ambiguous situations.
- One of their favorite questions to ask is "what if?"
- They are often tinkerers who enjoy getting into and seeing what makes something tick (this goes beyond software)
- They are often people who feel comfortable pushing at boundaries, and will often, if given the chance, go beyond the boundaries as seen or written
- Testers like to "poke the box", just to see what will happen.

In the following sub sections, I have included "Letters to Me" from a few software testers that have examined their careers and made some considerations of what it is that interested them when they were in their teenage and young adult years, and how these revelations helped them consider and understand why they might want to consider being software testers as a career.

[Editors Note; I had hoped I'd be able to link the Letters to Me to this section, but so far, the way to do that has eluded me. If they can't be linked, I'll put them into Element Boxes and include them here. Downside, it will make this section inordinately long.]

CHAPTER 8 Personal Perspectives on Software Testing

Chapter Outline

8.1 If I Could Write a Letter to Me: Michael Larsen

If I Could Write a Letter to Me

By Michael Larsen (mkltesthead.com)

There's a famous song in the country music world called "Letter to Me", written by Brad Paisley. In it, he writes from the view of his adult self to the boy he was when he was 17. In that letter he offers some advice to himself back then, much of it having to do with life, love and opportunities he'll wish he hadn't missed.

It's from this idea that I wrote a letter to myself as a teenager, so I could suggest to myself some attributes that might help me know if I'm a good candidate for software testing.

Dear Teenaged Me,

I know that most of your time right now is spent realizing that you are semi-photogenic, have a decent singing voice, and you really like taking that Fender Jazzmaster down to bare metal and building it back up. I also know that you like twinkling around on that beige putty colored Macintosh that your parents set up in your brother's room because they can actually get into it (unlike your room), but seriously, I want to talk to you about something.

At some point, you are going to put all your energies into being a musician. It's going to be a fun ride, it's going to impoverish you, and at some point, you are going to have to get down to brass tacks and dig your way out of a financial hole. When that day comes, you are going to find yourself surrounded by a strange breed of people, a wonderful breed. They are inquisitive, adventurous, creative and do some pretty cool things with other people's software. These people are called "testers" and believe it or not, you have a lot of the attributes they'd be looking for.

First of all, let's take a look at that guitar that you have taken down to its bits countless times. Why do you do it? I'm guessing because you want to see what makes it tick, and what changes and improvements you can make. Maybe it's just a cleaning or removing scratchiness from potentiometers, but the fact that you thought to break it down to look already points to you having the temperament of a tester.

Check out your reading material. You like to nerd out on unusual topics. You get a kick out of history. You enjoy philosophy. You actually own a book called "Words From the Myths" because you had to dig in and understand why words like "Hubris", "Nemesis", "Chaos", "Cosmos" and "Sisyphean" meant what they did. You aren't really content with people telling you something is true, you needed to find out

People are often defined by who they hang out with. You are hanging out with a bunch of musicians right now, and while you may think that that's all that you are, you're wrong. Many of you are drawn together because you share similar outlooks and ways of looking at the world. Among you are a CAD designer, an inveterate tinkerer and a rock solid network administrator, and that's just counting the guys in your first band. You don't realize that these are your future band mates careers, but if you were to look at them for any length of time, it's self explanatory. But there's something that sets you a little bit apart. You like to argue why things work or don't work. You have a passion for doing things right. Not that they don't, but you come at it from a different angle. While they are focused on the actual building of things, you tend to look for the structural weaknesses and what could go wrong.

You're not a pessimist. . . you're a tester!

Can you count the hours that you and your band mates looked for the optimal way to load and unload your gear, wire up your racks, maximize your wireless gear, and all of the other "refinements" you made while you were playing shows? Testing, my good friend.

I know that a lot of this might seem a little weird, and of course, if I were to say that the reason you ultimately decided to go into testing was entirely temperament, I'd be lying to you. Those all pointed the way, but the real reason you decided to make it your life's work was the people that were part of it. The temperament got you through the door. Your compatriots in arms are the ones that convinced you to stay. You owe a lot to them, too.

I'm going to close this for now, but I just wanted to give you some things to consider as you are plying your trade as a musician and trying to figure out what you want to be when you grow up. I'm not so sure I'm a good authority on the growing up part (by the way, I'm sorry for the broken ankle, lacerated wrist, splintered shoulder tuberosity and broken tibia and fibula you'll have to go through) but seriously, consider the testing avenue. You might be amazed at how much like "home" it feels when you get there.

Sincerely,

The Adult version of you (grown up? Not so much)

8.2 If I Could Write a Letter To Me: Ben Simo

When I was 17:

My primary interests (other than girls) were photography and computers.

I enjoyed programming.

My mother didn't value my "playing around" with computers.

I didn't quite understand that software development was a profitable profession separate from the engineering geniuses who designed the machines.

I certainly didn't know that one could make a living testing software.

I was a bit of a software pirate. I didn't care much about intellectual property and copyrights.

After some time doing a variety of odd jobs (including work as a part-time database developer) I fell into testing at the young age of 21. I was hired as a "data collector" to execute test cases and report results. I quickly learned that the test cases were ambiguous and investigation was required for good validation and verification work. I learned that testing was investigation. I liked trying things out. I liked the experimentation. I liked discovering important things. By the end of that project, the optimistic engineers who thought they could pre-script everything and only report pass/fail results were fired (contract not renewed) and I – the lowest paid employee on the project – was running things.

My fall into testing also included falling into TACO2 (Tactical Communications Protocol, v2). I became the expert on TACO2 performance. I provided tech support and testing services for TACO2 users for many years. I thought of myself as more of a TACO2 expert than as a tester. Even though I worked for a testing organization and the majority of my work was testing, I didn't identify myself as a tester – because I didn't realize testing was a skill of its own. People encouraged me to go back to school and get a degree in Electrical Engineering so I could be a real engineer instead of an "Engineering Technician" or "Functional Analyst" (the two titles I carried during most of my government testing work). I wasn't told that testing could be a career. Although my work for dozens of companies introduced me to many contexts, most of my work was with programmers or users. I didn't encounter many people in explicit QA or testing roles.

However, I often fell into testing roles that had nothing to do with my TACO2 expertise. I was brought into projects to help define test approaches and communicate results with stakeholders for technologies I knew little about.

When I left government work, I was given the title "QA Engineer". I read a bunch of books from people telling me it was my job to engineer and enforce process that would produce quality software.

I got arrogant.

I made enemies.

I learned the hard way that software development and testing is at least as much a social activity as it is a technical activity.

I began to better understand the importance of context.

Looking back, I'd want to tell my 17 year old self (and other young people considering QA/testing as a career) that:

- Testing is important.
- Testing is a mindset and set of skills that differs from programming.
- Learn to program; it can help you be a better tester.
- Software development is a social activity. Don't just focus on the technology.
- Money and hard work goes into producing good software. If you expect to be paid for your work, pay others

for their work.
- Not everyone develops software in the same way. This is ok. There is no one best way. Adapt how you work to the context. (but don't be afraid to push people to try new things).
- Testing is context-dependent
- Testing skills are valued independent of your subject matter expertise.
- You can't test it all, so choose wisely.

–Ben

8.3 If I Could Write a Letter to Me: Timothy Western

My mind's been wandering today, and a thought came to me. Were there early warning signs that I could be a Software Tester or QA style professional? If so what behaviors, characteristics, or approaches have I used before I ever even thought about going down the Software Testing path?

I can't speak for everyone, but I know, for a lot of young people, it's HARD to figure out what you are good at, what you have skills for. I remember taking two or three different career tests, from the ASVAB, PSAT, PLAN, whatever. One of them even broke your skills and interests into little slivers of a circle; in my case I had one that barely bordered engineering, but others that included the medical field. At one time I contemplated a career in medicine, believe it or not, but my strong math skills I felt would go to waste there, and well I couldn't resolve the idea of being around Sick people all the time. Oh yeah, but the rub is that now I work with sick computers or sick software all the time. Ask me now if I knew that which I'd choose? It's not an easy answer.

What helped me make my choices were trying to figure out what subtle skills I had that fit into various fields.

- Programming seemed a winner as part of a career path because my mind seemed well organized for it, and it was interesting, until I found out 90% of the Enterprise stack is essentially a rehash of previous Enterprise software :/ Or it surely seemed that way after a while.

- As a Freshman at College, I broke my glasses right before a retreat. When I returned home the next weekend I went to pick out a new frame exactly like it so I could just transfer the lenses in place, and I found one just like it. Except, I remarked, this frame isn't the same size as the others. The clerk swore up and down that wasn't likely, but after i tried them on, looked at them, held them together I was convinced in my observation. She went and looked the numbers up. Sure enough, it wasn't just a style # difference, it was about 3-5 millimeters smaller. Call that an early warning sign, as I often observe things that others in their normal daily life either ignore, or are oblivious to their existence.

-Tim