# Transcript: IT Podcast - The Art od Debugging w Carolyn DeCusatis and Mark Kenkead

*The following transcript is a verbatim account of the video or audio file accompanying this transcript.*

Speaker #1 (Narrator):

WGU's IT audio series flexible, portable, profound.

Speaker #2 (Carolyn DeCusatis):

Hello. I'm Carolyn DeCusatis.

Speaker #3 (Mark Kenkead):

I'm Mark Kenkead.

Speaker #2 (Carolyn DeCusatis):

We are some course instructors on Software team. We were talking about how the other day I was eating with a studentwho was in one of our later courses, close to graduation. He told me wistfully, "I thought that when I was getting close to graduation, I would be so fluent in Java that everything that I wrote would run the first time. All that I would have to do would be to type it in and it would all go.'' Well, that hasn't been my experience and I teach Java.

Speaker #3 (Mark Kenkead):

Yeah, certainly not. That's like wishing for a miracle. I think once I compiled a file and it compiled the first time, back when you didn't have the IDE yelling at you, telling you what to do. I was so shocked that I had to go back and reviewthe whole thing because I was looking for the error that I missed. It's virtually impossible it seems to me to ever have the expectation that the whole program's going to work, far less just even the little bit of code you write.

Speaker #2 (Carolyn DeCusatis):

Basically, when you are a programmer, a lot of what you're going to be doing is debugging your code and debugging other people's code. If you're going to do that, we should talk about how you do that. How to do it effectively. We pulled an article that discussed that: "7Steps to Debug Efficiently and Effectively" by Matt Goldspink, and we wanted to talk about it a little.

Speaker #3 (Mark Kenkead):

Why don't we just dive into reason number one, or step number one, I guess it is. How to debug your code efficiently and effectively? Number one, he sites, is always reproduce the bug before you start changing code. He says, "So many times, I've seen the description of a problem and immediately make random assumptions and proceed to make changes.'' I think we're all pretty guilty of that, don't you think?

Speaker #2 (Carolyn DeCusatis):

Oh yeah. In our environment, what I do a lot of the time is, when working with students who have gotten back and evaluate a report, spend some time with them trying to reproduce the bug that the evaluator found. Because a lot of students wouldn't, I mean nobody submits the code saying, ''It's not going to run the first time. I've debugged this. This is my code. ''Yet fresh set of eyes, they found something. So the first step is that hunting on what did they find, before we can fix that.

Speaker #3 (Mark Kenkead):

Also, another case is user error.It possibly could be a user error and so you want to make sure you can create the bug. Otherwise, you're wasting your time hunting something that may not exist, or maybe it was caused by something else they were doing. Those are really hard to track down, and the more complex your program gets, the harder it's going to be to find stuff sometimes. You definitely need to make sure you know exactly what it is. Sometimes getting the user to present a test case to you when you're in the professional setting, shall we say, not just trying to get your thing evaluated, it's really important to have a workable test case that breaks your program. You solve it so much faster.

Speaker #2 (Carolyn DeCusatis):

It's also good to have the User-centered Design Process, that gives you a little bit of feedback on how users can actually use your user interface. Because they won't have coming at it with the same assumptions that you have.

Speaker #3 (Mark Kenkead):

We can move on to number two, which is a biggie for us in the Java world. This is one of the big things that people encounter right away and they don't understand, but they need to, is understanding the stack traces. The errors that you get at the bottom of your IDE when something bad is going on.

Speaker #2 (Carolyn DeCusatis):

Now, I support the course on Mobile Application Development. One of the early things that we have to do is explain to students where they're going to find these stack traces. Because in the mobile course, you have to look in something called the Logcat, not in the terminal, and that's not intuitive. But they're so valuable at figuring out why your program just crashed.

Speaker #3 (Mark Kenkead):

It's useful because it tells you exactly the line that the crash occurred. The reason why they look so ugly is because they're giving you the ability to forensically look through every method that was called, and getting down to the very last thing that actually caused the explosion. Because sometimes it's something internally in Java that would not get normally triggered with your code, but you passed it something bad and so you're able to look at that. Those are good things to really learn how to read and filter out the noise and find out where the real problem is.

We can move on to number three. Number three is, write a test case that reproduces the bug. I think this is related to number one. I disagree with that. This is a very important thing to do if you think you can review the code and see what's going on wrong. But it goes back to number one where you really should have a repeatable case and be able to force it to break at your will so that you can then figure out exactly what's going on. What do you think of that?

Speaker #2 (Carolyn DeCusatis):

I think it's also thinking about a multi-developer environment, that you don't want other people to run into the same problem you had, so you might want to have a JUnit test done and once you've discovered it, it's an issue.

Speaker #3 (Mark Kenkead):

Four is know your error codes. This is web-related here, they're talking about something. I think this one goes back to the stack traces, which is know how to recognize when you're getting a NullPointer Exception, what does that mean? When you're getting a number format exception. How could that happen? As you get used to looking at the stack traces, you get to learn what these errors are. In fact, you should experiment in your code, I think, and break stuff and come and figure out what does it look like when I do this?

Speaker #2 (Carolyn DeCusatis):

Well, I think they'll be lucky if they have to break stuff on purpose. I think it's going to happen one way or another.

Speaker #3 (Mark Kenkead):

Well, it's a dream that you're the only one causing the bugs. Here's a good one. It says Google, Bing, Duck Duck, Go. Looking up the errors that you see commonly, your compiler, your output will have something that looks like a compiler error, or system error. If you're really confused about it, looking it up at Stack Overflow or one ofthese other places can help you find the answer.

Speaker #2 (Carolyn DeCusatis):

Yeah, Stack Overflow is your friend, it's filled with your friends.

Speaker #3 (Mark Kenkead):

Now sometimes your friends don't actually have the right answers. So you have to be really careful and make sure that that answer makes sense or it actually leads to something that's worthwhile to do, or fix. Then we go on to, it says,''Pair program your way out of it.'' What do you think this means, Carol?

Speaker #2 (Carolyn DeCusatis):

Well, in the context of our school,it means make an appointment with a course instructor to share screens and try and debug the problem together. But in general, having two people working together is more efficient than one. I mean, when you go into XP, you go into all the agile methodologies, peer programming was shown to be very valuable.

Speaker #3 (Mark Kenkead):

Yeah, and that's why the evaluators will find something that you haven't looked at. Because it's possible we all get, it's a very natural thing, very tunnel focused on the problem and we miss test cases, miss pathways to code. That's when things appear. Then celebrate your fix, which means I guess go back to the next bug and start working on that one, because they are always there. There was one other thing I wanted to point out in here. The first comment here is the guy said, ''Use a debugger,'' Which you should learn how to use the debuggers. Then of course, check your environment, make sure your environment is set up properly. Do you have any closing thoughts Carolyn?

Speaker #2 (Carolyn DeCusatis):

Happy coding.

Speaker #3 (Mark Kenkead):

All right, happy coding. Take care and we'll talk to you next time.

Speaker #1 (Narrator):

Schedule time with your course instructor to explore more deeply. WGU, a new kind of you.