

## Re: How did you learn your debugging skills?

**Source:** <http://coding.derkeiler.com/Archive/General/comp.programming/2004-06/1596.html>

---

**From:** David (*FlyLikeAnEagle\_at\_United.Com*)

**Date:** 06/28/04

Date: Mon, 28 Jun 2004 12:23:10 GMT

Hi Esther,

On Mon, 28 Jun 2004 01:44:42 UTC, esther@bitranch.com (Esther Schindler) wrote:

> *I'm writing my quarterly column for Software Test & Performance*  
> *magazine (those fine folks at <http://stpmag.com>), this time about*  
> *"best practices" in debugging. Okay, okay, I'm not keen on the "best*  
> *practices" term either; it sounds like the sort of expression used by*  
> *middle managers who jingle the change in their pockets. What I'm going*  
> *for, though, is the "rules you live by" when it comes to debugging.*  
> *The essential guidance. That is, after a long frustrating weekend*  
> *spent looking for a bug, when you finally find it you say to yourself,*  
> *"I'm such an idiot. If only I'd remembered [xyz], I'd have found that*  
> *defect a lot sooner." What's XYZ, for you?*

These days, after twenty plus years playing with computers, my mistakes are generally far between. There are still mistakes to watch out for. Most of them are related to thinking that I understand what a relatively new system is going to do without having had the experience to actually trust the system to perform as advertised. Systems such as Microsoft Windows change with each release and often do not behave the same. Subtle changes in behavior can result.

Like many developers, I maintain code written by others as well as what I have contributed to the project. As such, my skills include working with other people's work. My most common experience is attempting to reuse or rework some portion of code written by someone else only to find that the code is not quite acceptable (I'm being very generous here) or just plain didn't work. After all, some programs appear to work correctly but actually don't. They will fail under their own use or especially when used for an unforeseen purpose. Consider a locking mechanism (if one was actually used) it may appear to work when used in an expected manner. Does it fail when stressed or when new features are added? I spend a fair amount of time rewriting others code to an acceptable level.

## comp.programming: Re: How did you learn your debugging skills?

Debugging is usually described as removing problems from the system. It may also cover proving that the system will work correctly and actually does.

The developer must understand that they are responsible for what they create. It is rare in most applications to trace failures back to their origins. However, just because you may not meet the person that swears at your application, cannot trust its results, loses an investment due to your failed code, or dies as a result of your code -- you may have to think about that at some point.

I'll add a little anecdote for you. While working for a large computing company, I had to correct some operating system flaws that were affecting a large credit card processing company. I was at the customer site observing the behavior and another colleague wanted to explain what his part of the system was responsible for. He said it was a storage system and wasn't fully functional yet. They were able to copy files in/out of the archival system but he explained that the 'dir' command wasn't working yet. Anyway, a few hours later, just after lunch when 200 or so people were hard at work, he was talking through the operation to another person. Well, he was asked how to find what was on a given disk and responded by leaning over and typing 'dir'. The system didn't respond. A few seconds later, about 200 people yelled that the system was down again, followed by an announcement for people to take a break while the systems were fixed and rebooted. I could tell he knew exactly what he'd done. The person he was explaining it to, a manager for the customer, didn't know he was responsible, but did say the loss to the company was about \$1M per minute outage and he'd better get it fixed yesterday.

> *A related question is... how did you learn the debugging skills you  
> have? Are you self-taught, or did you learn in school or by some sort  
> of apprenticeship? How do you improve your debugging skills, and in  
> what way do you judge how good you are at finding bugs? And coming  
> back around to that "best practices" stuff: when you show younger  
> programmers how to debug, what are the techniques you're most likely  
> to teach them?*

I'm mostly self taught in terms of how to diagnose problems. We all have our own internal understanding of what our/the program and system are trying to do and what it did that surprised us. Debugging for the new programmer is generally an exercise in trying to discover why their code didn't perform as expected. Upon finding the problem, they learn of their mistake, perhaps how to fix it, and hopefully how not to repeat that procedure in the future. As we get better a few rules may be added such as correcting mistakes

Re: How did you learn your debugging skills?

comp.programming: Re: How did you learn your debugging skills?

that we find we have done before.

My first programming was with machine code on an 8008 with a front panel. It was at an older friends house. I was working my way through understanding electronics and getting several amateur radio licenses. By high school we had an H-P 2000/A BASIC system. It caught line typos pretty much immediately. The rest were logic flaws that had to learn on my own. I was taking college computing courses by my junior year in high school. I'd like to credit the formal teachers I had then who helped guide everyone through learning how to program. They had an IBM 370 with Assembler, COBOL, FORTRAN, and JCL on it. The rules for each exercise were simple – you had three tries to make it though the compile, link, and run phase. There was no fourth try, just a failing grade for that assignment. I never found that a problem, but many students did. While it was necessitated by the computing systems they had, it helped many people understand that they need to think about their tasks up front and examine their code before running it.

The software development effort is not all that different from a task such as writing an article that is to be published. You get a certain amount of time and effort and experience leading up to releasing/publishing the work. After that, you learn whether or not the intended audience understands what you wrote. A few spelling errors can cause a little grief, writing an untruth can get you into trouble, and so on.

- > *If I may quote you, please let me know your real name, company, and*
- > *title, even if it's by personal e-mail (I'm at esther@bitranch.com).*
- >
- > *Esther Schindler*
- > *contributing editor, SD Times and Software Test & Performance*

David