Do you think Debugging (System Troubleshooting) is one of the most important measurable technology-agnostic skills a programmer can have? [closed]

Asked 16 years, 2 months ago Modified 16 years, 1 month ago

Viewed 1k times

6

As it currently stands, this question is not a good fit for our Q&A format. We expect answers to be supported by facts, references, or expertise, but this question will likely solicit debate, arguments, polling, or extended discussion. If you feel that this question can be improved and possibly reopened, visit the help center for guidance.

Closed 12 years ago.

There are many skills a programmer could have (understanding the problem, asking good questions, good design skills, etc.).

I think **System Debugging skill** is incredibly valuable. This general skill of debuggin any technical system (from the batteries being dead in your remote control to signal interference from your neighbor's Ham Radio).

Here's the method I gave students when I taught Computer Programming:

1. Define the Problem (when I perform XYZ Repro Steps, I get ABC Symptom)
2. Identify the Testing Scope and break into sections.
3. Test each section using process of elimination to find the section causing the problem.
4. Break section down into subsections if needed.
5. Analyze the subsection causing the problem.
6. Fix.
7. Text by using Steps to Reproduce the Symptom.

Whadaya think?

debugging **language-agnostic**

Share

Improve this question

Follow

edited Nov 1, 2008 at 18:48

asked Oct 20, 2008 at 2:30

Clay Nichols

12.1k ● 30 ● 114 ● 175

This question belongs on some kind of blog... – Richard T Oct 20, 2008 at 3:05

In One Sentence "Divide and Conquer" – Priyanka Mishra Jul 20, 2009 at 11:44

9 Answers

Sorted by: Highest score (default)

4

if you define "troubleshooting" to include "debugging" then yes, it is critical!

EDIT: based on your edits, the top-down process-of-elimination technique you listed is systematic and very valuable.

Another technique is reasoning backwards:

- start at the end, with the variables/calls that produce the symptom/bug
- reason backwards from there - what could have caused this incorrect value or bad call?
- continue to trace backwards in the logic until you find the culprit

This is useful when there are a lot of forward-feeding paths/possibilities that could have caused the problem, but has the advantage of not needing a debugger or tracing to figure out the most likely cause.

Another technique is the usual-suspects technique, where you begin your investigation with whatever part of the code was touched last or has given you the most

problems, to see if something changed in it to cause the new issue

Another technique is to just sit and think about what situations could possibly produce the bug/behavior/value observed. This technique is useful when you're in a hurry and don't want to systematically scan a lot of places, but requires that you already have a thorough understanding of the system. This is useful when the bug/behavior in question is due to a design flaw or oversight and not a coding error.

Share Improve this answer Follow

edited Nov 3, 2008 at 14:34

answered Oct 20, 2008 at 2:46

Steven A. Lowe

61.1k ● 19 ● 135 ● 204

Thanks. Good clarification. I updated the question. – Clay Nichols Nov 1, 2008 at 18:43

@[MrAnalogy]: edited based on your edits – Steven A. Lowe Nov 3, 2008 at 14:34

2

Troubleshooting in the world of computers typically implies your basic "My-computer-doesn't-work-can-you-fix-it" type work that the guys behind the counter at Best Buy do. I think what you mean is Debugging, and while both can be valuable (I'm certainly glad I don't need to go running to support for every glitch my computer has)

Debugging is certainly the more critical skill for a developer. No matter what the language, knowing how to track, identify, and eliminate bugs is one of the single most difficult and most commonly performed tasks a programmer will do throughout his or her career.

That being said, I would actually say that what I consider to be the most important skill in my arsenal is perseverance. When my manager puts me on a task they know that I will get it done, no matter what. Period. No matter what roadblocks are put in my way, no matter what the timeline, no matter what the end goal, I will do as much as is humanly possible to meet their expectations until the job is done or they are satisfied with the results. That, by the way, is a very important distinction! The original goal is not always feasible, and when it's not it's my job to ensure that management knows exactly why it's not and what their alternatives are as soon as possible. And it's important to provide those alternatives. No one wants to hear, "Well, sorry. You're pretty much screwed." Providing a different solution to the problem is part of that perserverance thing. You never quit unless you're instructed to.

In short, the ability to see a project through to the end without getting distracted or giving up is, in my opinion, the single most effective thing you can do to set yourself apart and make yourself desirable to employers. And beleive me, it's damn hard! But that's what makes it valuable! :)

Share Improve this answer

Follow

answered Oct 20, 2008 at 3:06

Toji

34.5k • 22 • 108 • 119

2

Troubleshooting skill is critical. Usually a good troubleshooter has more questions than answers. They will also have a methodical approach to testing solutions to the problems they find. They will not fear what they don't know or be embarrassed to ask a "stupid" question. They learn quickly. All good traits for a programmer.

Share Improve this answer

Follow

answered Oct 20, 2008 at 3:16

Jibba

426 • 4 • 14

1

"Lazyness"

Lazy programmer search on Google

Share Improve this answer

Follow

answered Oct 20, 2008 at 3:05

Tim Jarvis

18.8k • 10 • 59 • 95

1

When I left college, I thought the key to being a great programmer was to make perfect (in both function and form) code the first time. Any problems that appeared were because I wasn't good enough, or because I wasn't



careful enough, or couldn't think far enough ahead, or spend enough time planning. If only I could work out all the problems on a whiteboard, I would be a great programmer.

I started my programming career working on a debugger. After a couple years of that, I was convinced that the most important skill for a successful programmer was debugging. I accepted that I couldn't write perfect code the first time, and neither could anyone else. Spending all my time on the whiteboard never produced working code, but it sure took a long time. Instead, I'd just write something that seemed plausible, and then debug. There was no way to write great code like I wished, but I could get really good at fixing broken code.

That worked. I was able to create shipping software, while only taking 2x or 3x as long as planned. I was able to fix almost all of the important bugs before shipping, without introducing too many other bugs along the way. (I wince to write that!).

Then I read Fowler's Refactoring. It completely changed my thinking. It taught me that, while I couldn't write simple, clear, clean, maintainable code the first time, I could get there. The reason I couldn't get my design right on the whiteboard the first time was because you can't tell if it's right until you see it in code. Only then can you decide what's wrong and how to fix it. The key seemed to be to get in to code as soon as possible, while making it

as cheap & safe as possible to fix the design of your code once it's written.

That lead me to Extreme Programming, where I learned that by writing well-factored code, where the classes are small, simple, and easy to understand, I could also have simple, easy-to-write, and easy-to-pass unit tests. When you code like that, a unit test failing points you to the root of the problem very quickly. When I do it right, tracking down problems is so easy that I rarely touch the debugger.

Now, I'm no angel of code. I rarely pair program, and I don't always write my unit tests first. I blame my parents. Oh, and the fact that my employer and my tools aren't as supportive as I could wish. And sure enough, I pay for it. Then I whip the debugger out, and apply those old debugging skills, while wishing I had just written better unit tests.

There's another place where debugging is critical: in production. When you're looking at a problem in the wild, being able to diagnose it quickly in a debugger is very powerful. I can learn to write programs that are good at self-diagnosis where I can predict weakness, but for the unpredictable problems, I think I'll always need a debugger.

answered Oct 20, 2008 at 5:15

Jay Bazuzi

46.4k ● 16 ● 115 ● 170

I think that commitment is more important.

0

Share Improve this answer

Follow

answered Oct 20, 2008 at 2:35

Hapkido

1,421 ● 1 ● 9 ● 13

0

It's important but troubleshooting/debugging for me is just:

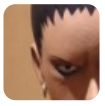
1. understanding the problem. knowing how the modules work together. OR knowing what you don't know about the problem/modules and;
2. asking the right questions
3. and proceeding with logical steps in solving the problem

So yeah, it's important, but it's not that far from the others you have mentioned. I'd measure (1) first. Present a problem with some parts intentionally left vague. He should be able to realize that. The next step is (2) he asked the right questions. And then attacks the problem non-randomly (3).

Share Improve this answer

Follow

answered Oct 20, 2008 at 2:59



moogs

8,192 ● 8 ● 47 ● 60

0

Knowing how to:

- Learn
- Research
- Accept you do not know everything
- Apply knowledge from past experiences to your current problem

- And, most importantly, explore alternatives to a problem you think you may have solved. You might come across a better solution.

If that is what troubleshooting is, then yes it is very important :D

Share Improve this answer

Follow

answered Oct 20, 2008 at 3:03

user19302

0

I like how one of my former bosses defined the 2 parts to being a developer that explains why a simple "Yes" may not be sufficient here:

1) Programming - being able to design and write code.

2) Troubleshooting - debugging, investigating issues to determine if it is a code, data, or network issue, fix bugs.

Share Improve this answer

Follow

answered Oct 20, 2008 at 16:10

JB King

11.9k ● 4 ● 40 ● 49

> And I guess my point is that if you can't Debug your own code, it's very very difficult to be a productive programmer. (And if you one can't debug others' code then one will be very s l o w at deugging even ones' own code.) – Clay Nichols Nov 1, 2008 at 18:47