Good Developers Do This

Last updated by | Joe Walling | 31 Jul 2024 at 17:58 GMT+4

A good developer isn't simply looking at code and making fixes. A good developer understands the problem they are trying to solve and how the application is currently solving that problem. You may notice that I often start my specs, tasks, and work items with a short description of the problem or what the customer wants to achieve. I intentionally do this in case when you are looking at the code you can think of a better way to do this than I'm proposing.

If you ask Aleksey, he'll tell you that he often challenges my spec based on what we are trying to achieve. By having a good debate, when appropriate, we end up with a better application. So don't just build something that doesn't seem like it will make our customers' lives better. When I'm writing a spec, I'm not in the code looking at how it is currently done, and you may have a better approach based on the existing code.

When you are working on a screen, you should understand a little about how it works so the requested changes make sense. If you are new to the application and the industry, it might help you to read the appropriate documentation in the user guide at Getting Started Archives - Dump Truck Dispatcher. You can access it and a chatbot by clicking the lifering at the top of the application and selecting "Online Documentation and videos.

Here's a few more descriptive phrases of what good developers do:

- Understand the performance and security ramifications of what they are doing.
- Always think about the end user and how you can make the usage of the product clearer and easier for them.
- Write unit tests where appropriate.
- Understand the impact of changes on other areas of code. For example, there are areas of code that have multiple related settings. Will changing one of the settings impact the others?
- Asks appropriate questions and isn't afraid to challenge a request that doesn't make sense. It may be that their knowledge of the product or industry isn't deep enough to understand, or it could be that the person that wrote the spec doesn't know something about the code that they know.
- They don't ask questions that they could have looked up in the documentation. That wastes their time and that of the person that responds.
- Reuse and refactor code. If they find themselves about to cut and paste, they ask the question of how they could do this differently.
- They pay attention to detail.
  They seldom have their code returned. It is better to get it right the first time. Taking a little more time upfront is much better than getting it done quickly and then spending several days back and forth on the PRs.
- They don't change things they don't understand.

I have seen numerous PRs returned multiple times, sometimes for the same thing mentioned in the comments. And often, these waits leave you with no more work.

When you receive a comment, don't limit your review and change to the single line referenced. See if that comment might apply to other places in your code for this PR. A code review isn't meant to catch every possible problem with your code. The code is your responsibility, not the reviewers. Use these reviews as an opportunity to improve.

I could go on and on, but I think you get the idea. These are a few things to think about in order to make this product and your career a success. As you show your understanding of the application and our processes, you'll be given more responsibility and raises.