DEVELOPER SKILLS
BECOME A BETTER DEVELOPER BY ASKING QUESTIONS
www.stevejgordon.co.uk | @stevejgordon

HOW TO BECOME A BETTER DEVELOPER BY ASKING QUESTIONS

13th January 2021 (https://www.stevejgordon.co.uk/how-to-become-a-better-developer-by-asking-questions) Steve Gordon (https://www.stevejgordon.co.uk/author/stevejgordon) Software Development Skills (https://www.stevejgordon.co.uk/category/software-development-skills)

It's the beginning of a new year, so I wanted to open with a post that I've been planning to write for some time but never quite gotten around to creating. I recently started a new job, joining Elastic (https://www.elastic.co/), to work on their .NET language client. It, therefore, felt appropriate to finally get this topic written and published. It's turned out to be one of my longer posts as there's a lot I wanted to say. I recommend grabbing a cup of tea or coffee and settling down to read this one!

ASKING QUESTIONS IS A POSITIVE ACTION

In my years as a software engineer, I've asked a lot of questions. I've also been approached by developers with their own questions. At times, those conversations have started along the lines of "Sorry to waste your time with something so simple…". In this post, I want to articulate why no developer, new or experienced, should apologise or hold back from asking questions. They are a critical part of building your own experience and becoming a better developer.

There is an almost infinite amount of knowledge to be learned. No single person can know everything. In software development, our industry moves extremely quickly. The latest new technology soon gets replaced by something else. The once preferred architectural style goes out of fashion or gets superseded by a superior pattern. In this aspect, all software developers face the same challenge of trying to keep pace. It's honestly impossible to truly keep up with everything.

Of course, one solution is to find a single technology and coding patterns, and then stick with them. This is not uncommon in large enterprises where the pace of change is often slower. Still, it's impractical to believe that everyone in the enterprise can know everything about their existing development environment. It also doesn't avoid changes in related technology and tools. Perhaps you use a source control product or planning tool, which will evolve over time, forcing some degree of learning and adaption. A once preferred API may be deprecated due to security concerns, forcing developers to understand and utilise the more secure alternative.

Software development is a craft, and like any craft, there are personal styles and choices to be made along the way. It's perfectly reasonable to wonder whether the code you've written is the "best" way to tackle a problem and for you to want to seek opinions from others.

Perhaps you're reading this as you are about to start your first software development job. If so, congratulations! Or maybe you started a new job in the last year. My principal advice is to don't be afraid to ask questions and don't feel that it makes you a worse developer. I'd happily argue that I consider someone with a questioning mind to be a better developer in some respects. It shows you're interested, willing to learn and keen to improve at what you do. There is nothing wrong with that!

For more experienced developers; from those with a few years of on-the-job experience to those who have been coding for decades, you should still have questions. I've observed situations where long time developers fear asking questions, perhaps even more so than junior developers. There may be many factors at play, but in part, I suspect it's a degree of imposter syndrome. A more senior developer may not want to appear naive in some aspect of software development. Perhaps they assume they should know something and are afraid to reveal that they do not. This can be quite dangerous as those developers may prefer to guess, rather than truly understand whether a choice is good or bad. To the senior developers reading this post, I ask you now, can you remember when you last asked a technical question at work. How many questions did you raise last week? If you struggle to answer these, then perhaps you are subconsciously avoiding asking questions.

Senior developers also change jobs, and while you may be in a senior position, it's reasonable to assume you will have many questions in your first weeks and months working for a new employer. You may be unsure why a particular approach is used, a specific pattern preferred, or what a specific piece of code does. In that case, there should be nothing wrong with asking. If the employer and managers frown on such questions, then perhaps they are not the best employer to be working for. It's unreasonable and impractical to assume that all new senior hires will turn up on day one knowing everything.

As a developer, your questions need not be limited to purely code-related topics. You may wonder why a particular planning process is used, how you are expected to balance your time or how to file an expense claim. These are also perfectly reasonable topics to raise questions about.

If you've read this far, you'll have hopefully picked up by now, I want to encourage everyone to ask questions. For it is the act of asking questions which moves our knowledge forward.

PREPARING TO ASK A QUESTION

It's essential to start by stating that not all questions necessarily need to be asked in person. For most problems, I'd advocate that you start by trying to answer the question yourself. By this, I mean, research the question online, in the company wiki, or in reference materials and documentation. It's rare that you'll be the first person to have a particular question and so the quickest way to get answers is often to Google or Bing for one first. Googling is still asking a question!

If your question is about some code that you're trying to understand, give yourself time to properly read (and re-read) the code. Also, ensure that you look at any tests. Some code can be quite complex, particularly in large, legacy codebases. While the code itself may be challenging to interpret, the tests, if present, may help you understand what it's intended to do. Reading and interpreting code is an essential skill for software developers to master which I've previously blogged about (https://www.stevejgordon.co.uk/become-a-better-developer-by-reading-source-code). Any opportunity to practice this should be embraced. It's okay if you still need to ask for someone's help after attempting to glean what the code does yourself, but start there first.

If your question is about why some code you've written is failing or causing an exception, spend a little time trying to isolate and understand the issue before engaging others. Check your code for errors that may be the cause. Did you write any tests and could that help prove which part of the code is the culprit? Have you debugged the application to track down where things go off the rails? Have you tried to reproduce the error in a smaller sample? These techniques are a great practice to get into. Again, the idea here is not to avoid asking a question, simply to ensure that the question you have cannot be answered in your own mind. As you progress in your career, you may solve more and more of your own blockers, but they need to ask for help will never completely go away. Sometimes we just have a mental block and asking for a second opinion is a perfectly valid way to prevent yourself spiraling into unproductivity.

ASKING QUESTIONS ON GITHUB AND STACK OVERFLOW

Also, consider that some questions may also be better suited to online forums and resources such as Stack Overflow (http://stackoverflow.com/) or GitHub (https://github.com/). Some questions are specific to a piece of failing code, ask for opinions on acceptable use of patterns or ways to use an open source library. For these, you may find that the broader online developer community is better placed to assist you.

Asking questions online can be quite daunting, even more so than reaching out to people you know. One of its advantages is that you'll open yourself to wider and more diverse opinions and ideas. It requires a little more filtering and personal judgment on your side, to validate that the answer seems reasonable. If, after receiving some answers, you're still unsure about them, you can always ask someone you trust for final validation.

Online forums support the development of good question-asking behaviour. For example, suppose you're opening a GitHub issue or starting a discussion about the correct usage of an open source project. In that case, you should attempt to seek whether it has previously been asked and answered. It can be frustrating for maintainers when a simple search would have produced the answer you needed. Failing to check can make the maintainer feel as though you are giving higher value to your time over their own.

You should also ensure you follow my advice above. If you believe you have found a bug or are struggling with the usage of a library, demonstrate and share a reasonable minimal reproduction rather than expecting others to read extra code or try to reproduce it themselves. Make sure you've attempted debugging your code first and read the documentation. If you're stuck after making reasonable efforts to answer the question yourself, don't feel afraid to post it. Jon Skeet has some great content that goes more in-depth into these practices and can also lead to you solving your own problem in the process. In particular, his question asking checklist (https://codeblog.jonskeet.uk/2012/11/24/stack-overflow-question-checklist/) and his thoughts on Stack Overflow culture (https://codeblog.jonskeet.uk/2018/03/17/stack-overflow-culture/) are both worth a read.

I've personally had mixed experiences on Stack Overflow. The culture there can be quite officious, and some members are a little too quick to criticise the way a question is worded or if it may be a duplicate. As long as you've made reasonable efforts to search for a previous answer and have taken the time to form a well-crafted question with code samples, try not to worry about the occasional lousy response.

Through this blog, I receive quite a large volume of questions via comments or the contact form. I'll admit, I'm not always great at keeping up. Sometimes I find that the question is too broad and would require too much time to fully answer. Other times I'm not even sure what the person really needs help with due to confusing phrasing or too many points being raised.

When preparing a question, put yourself in the position of the recipient. Assume no prior knowledge about your problem domain and determine what the core information which you'd expect to be able to understand the question is. Focus on forming a specific and concise question that will not take a tremendous amount of time to interpret and answer. I've had emails where the person simply wants to ask too much in one go, and I have no idea what's the critical thing they need to understand to unblock them. I've also had emails where the person would like me to take a one-liner requirement and essentially explain how to build an entire application. That's far too broad for me to answer in my personal time.

I'm not trying to put you off asking questions in my previous paragraph. I want to remind you that the person on the other end is human and has their own work and commitments. Take that into account and ensure that you don't unfairly monopolise someone's time with vague questions.

ASKING COLLEAGUES

Some questions will be naturally more specific to your place of employment and its product. You may wonder why a particular piece of code is used in preference over another. You may be unsure how a feature in a software product actually works. Those may have been asked and answered in a company wiki. Still, in many cases, the answers may never have been documented. In those situations, you will need to track down someone to ask in person, or via chat. You may also be surprised to learn that more senior colleagues are also unsure of the answer. This is not rare as knowledge drifts away over time. The code you're working on may have been written a long time ago, and decisions made back then may no longer be apparent.

I firmly believe that there are no stupid questions. Everyone's mind works in different ways, and what may seem obvious to one person, may be less clear to another. I prefer it when someone asks a well-formulated question and shows some sign of having tried to answer it themselves. Try to form your questions succinctly and clearly so that the person you ask can quickly work out what you want to understand. Also, be sure to include any relevant information you already know or have found before asking the question.

Rather than "What does this piece of code do?", favour something more along the lines of "I'm trying to understand what this code does. I've Googled, and I can see that it might be the XYZ pattern, but I'm still unclear on what it's really doing. There are no tests for the code, so I'm unable to confirm the expected behaviour. Can you help?"

A question that shows that you've spent time on the basics, tried to investigate it yourself, and have prepared your thoughts is always preferred. Sometimes, you'll be under time pressure, or perhaps you're worried you've broken something crucial. It's fine to use your own sense of when it's best to raise something early, rather than lose precious time trying to answer it yourself.

Sometimes you may be able to fully answer the question based on research, and that's fantastic. There is also nothing wrong with asking someone simply to validate what you believe the answer to be. "I'm trying to understand this code, I believe it does X, and I think it's using this pattern because of Y. Is my understanding correct?" is a perfectly valid question.

HOW TO ASK QUESTIONS

Once you have tried to answer the question yourself and if you still need some help, you may need to ask someone the question directly. Your first choice is who may be best placed to answer the question. Try to direct your query to the most appropriate person. It's not always easy to know who to ask, so it's reasonable to reach out to someone you know or work with to ask for advice on who to speak to.

Once you have identified a person to talk to, consider whether you need to speak face-to-face, or if other methods will be better suited. Your choice here will depend on a few factors such as, how complex is your question? How urgent is it? How does the person prefer to be engaged? How busy is that person?

As a general rule, I recommend asking most questions via your internal chat system. Most companies use some asynchronous messaging service such as Slack or Microsoft Teams. The advantage of these technologies is that you avoid interrupting the flow of the person you are speaking with. They can focus on their work until they have a moment to check and respond to chat messages.

Some questions may be easy to ask via chat, and you can receive an answer via the same medium. Some will be hard to explain or likely require some back and forth discussion. Those may end up as face-to-face conversations. To begin with, consider that if it's hard to write your question in a chat message because it is complicated. Perhaps you need to simplify it first.

Rubber Ducking

As a slight aside, I've previously talked about Rubber Duck development on this blog (https://www.stevejgordon.co.uk/do-we-have-an-obsession-with-ducks-in-software-development/an-obsession-with-ducks). Sometimes, in the act of explaining your problem when asking a question, the answer will leap out at you. Our brains can work in mysterious ways, and sometimes, we need to switch gears to unblock ourselves. One of the advantages then of favouring chat messages over face-to-face is that you may answer your own question while writing your message. In this case, not only do you get an answer, but you may never even interrupt the other person. They may have helped you without ever being aware of it. This is a real win-win, in my opinion.

WHEN QUESTIONS ARE TOO COMPLEX FOR CHAT

Chat is an excellent place to start, but it won't suit every question as I've mentioned. The person you are asking may realise this and ask you to meet in person or over video conferencing. Sometimes, you may discover it needs to be face-to-face in the first instance. If so, I recommend avoiding starting the engagement since such interruptions can really ruin someone's flow. In this situation, I recommend using your workplace chat to message your recipient to ask for a moment of their time when they are next free. That way, they can focus on finishing their current task before responding. As a developer, I prefer when someone asks me a question in this manner since interruptions when coding can be a real productivity killer if they occur at a crucial moment in some coding activity.

There will always be odd exceptions to this advice. If you think you've just brought down production, speak to someone immediately!

In cases where you have more general questions, consider if you have a scheduled meeting that may be a good forum. Sometimes raising a question at a meeting can provide more diverse opinions and allow everyone to benefit from the discussion. Don't be afraid to ask what feels like a fundamental question. If you have a question, someone else may do too. I've left meetings where a majority of the attendees actually didn't understand something crucial, but all were too afraid to ask, each coming up individually afterward.

SUMMARY

Whether you're studying to become a software developer, you're in your first day as a junior, or you're a senior developer in a long-term position, don't fear asking questions. Seeking knowledge is a good thing. It will help you grow and become a better developer. You'll more deeply understand your work and absorb ideas and reasoning from those around you.

Remember that while there are no stupid questions, there are badly formed ones. Make sure that your question is clear and as concise as possible. Share relevant information and before asking it, do your research and try to answer it yourself. If you still need help after this, don't be afraid to reach out to ask your question. We all need help from time to time.

STEVE GORDON (HTTPS://WWW.STEVEJGORDON.CO.UK/AUTHOR/STEVEJGORDON)

Steve Gordon is a Microsoft MVP, Pluralsight author, senior developer and community lead based in Brighton. He works for Madgex developing and supporting their data products built using .NET Core technologies. Steve is passionate about community and all things .NET related, having worked with ASP.NET for over 15 years. Steve is currently developing cloud-native services, using .NET Core, ASP.NET Core and Docker. He enjoys