Brain Book Buddy Boss Break: A Practical Debugging Framework for Learners

Mar Doig

Engineering Education Lead at Perk

Introduction: The Debugging Mountain & The Error Misconception

Harken your memory back to your first error message, however many years ago that might be. You likely felt a small jolt of panic, a worry you'd irreparably **broken something**. Nowadays, you hopefully see errors as the lane bumpers of your programming journey. Errors help us understand flaws in our original design, consequences of code changes on original code, and limitations of our system.

When teaching students at the beginning of their journey, it's important to remember they will require psychological coaching to reframe these errors from failure, to progress. As experts in their field, they rarely make mistakes in their work. In learning a new skill it's key to foster the growth mindset that's so fundamental in programming.

This growth mindset muscle must be strong, so strong in fact that it can conquer the usual process of fixing one error only to find another one. An experienced programmer would see this as progress, the stepping stones that will lead to their desired end result. Whereas students often see it as compounding failure, eroding away at their frustration tolerance. This can be a huge blocker in their learning and lead to demotivation or the classic "programming is just not for me" mindset.

This chapter will lay out the **5Bs framework**[11] **- Brain, Book, Buddy, Boss, Break -** a guide for students encountering debugging. This strategy codifies the organic path of problem solving that is cultivated in engineering. It will allow the student to develop a systematic approach to problem-solving that will keep them moving forward in their debugging, increasing their frustration tolerance as a result. Throughout this framework, we'll also explore how learners can thoughtfully leverage modern AI assistants as powerful aids, while still ensuring they build deep conceptual understanding and robust problem-solving skills.

A Note About Debugging and AI

However you feel about the broad usage of AI, their integration into learning environments demands our careful consideration. These powerful assistants can remarkably accelerate experienced developers and are invaluable for getting the job done efficiently. However, for the novice programmer focused on building foundational understanding and their own problem-solving abilities, they present a double-edged sword. If an AI consistently serves the perfectly cooked meal, the learner may miss the crucial, sometimes messy, process of learning to prepare it themselves. This may hinder the development of their core analytical and debugging muscles. With this in mind, the 5Bs framework presented in this chapter will explore how to thoughtfully integrate AI at each stage, aiming to strike a crucial balance: leveraging these incredible tools appropriately while always prioritizing and fostering deep, independent learning and critical thinking.

[11] We expand on the "Brain Book Buddy Boss" technique, which is used in classrooms, but seldom mentioned in teaching literature or books. We could not track down its exact origins, but in chapter 4 of Claxton (2018) it is introduced as a variant of "Try Three Before Me". "Try Three Before Me" guides pupils who got stuck to first try solving or troubleshooting their problem themselves in 3 different ways, before they escalate it to the teacher. "Brain Book Buddy Boss" is a more specific variant, which suggests specific stops on that troubleshooting journey.

BRAIN: Engage Your Mind Before Your Keyboard

This core step cannot be understated. Before engaging with any resource a student must build practice in thinking about their problem. They will likely be used to, from their current problem solving ability, creating a hypothesis and testing it to discover a result. It's likely this is their approach when writing code too. The key is to remind them it's the first port of call for an error message too.

In this part of the process, no AI should be used. The thinking process is precisely where the learning happens, and frustration tolerance is built. The struggle between the student and the problem will exercise their problem-solving muscle thanks to the resistance put up by the code. AI could give them the information straightaway, but this is not the goal.

Here are some practical techniques a student can engage:

1. **Read the error message:** This is a skill in itself. Knowing how to spot which file and line are causing the offending problem will give us a first place to start the investigation, and may even reveal it was all down to a simple misspelling or syntax error.
2. **Reproduce the error:** Does the bug happen if you run it again? The classic turn it off and on again still yields results.
3. **Identify the gap:** Note what you are getting and compare against what you were expecting. Is there an interesting relationship between these two pieces of data? Is the answer "off by one"?
4. **Formulate a hypothesis:** Turn the gap into a hypothesis, "I think X is happening because of Y."

At this point we are ready to test Y until we figure it out, or we find a new error and go back to the beginning.

5. **Simplify and isolate:** Look at what changed since the code worked. Try to isolate the problem and see if it's still reproducible. You can also comment out all code and uncomment it out slowly until the error reproduces, finding the source of your problems.
6. **Sample inputs and testing:** Change or hardcode a variable to test your hypothesis. This is where print debugging enters the scene too. Print key variable states before and after suspected problem areas, and remember to label them!
7. **Debugging:** Use debugging tools built-in to the IDE to set breakpoints, and step into functions.

This process is designed to be done in loops, until one moves forward or has to use another step in the framework. Encourage students to keep checkpointed versions of their work, as undo commands only provide a limited safety net.

Tip for Educators

Encourage the student's self-reflection during this process by prompting them with simple questions:

- What have you tried so far?
- What does the error say?
- What happens when you change this?

BOOK: Consult Your Knowledge Base

Official sources of knowledge should be the core of a student's debugging. These are presented in the order the student should consider consulting them.

Own Materials

A common first step is for the student to consult their learning materials, their notes or their code. This is usually a very useful resource, as they are deeply engaged with it and it's likely to have similar examples to the problem they're trying to solve. Students may be motivated to produce tidy, well-commented code in the course of their learning, when they know they're likely to go back to it when they're stuck.

Documentation

Despite it being a complex source to parse, I would always encourage the students to engage a primary source, the documentation. Getting comfortable reading technical documents is part of the learning process, and in most cases that's where the most authoritative answers will be found. Finding answers to a problem via reading documentation is a hard skill to master, but it is extremely valuable.

Books

While it isn't the quickest way to solve a bug, many engineers' favourite pathway into deeper understanding is reading published titles. Authors have taken much time and deliberation over every sentence in their book, and this leads to the highest quality explanations of concepts. Books are a form of linear learning, which contrast to the other options' fragmented learning. While a google search might fix a bug, reading a book chapter will fix the misunderstanding that caused it in the first place.

Reading around a topic when one is stuck can be a natural way to gain deep understanding without feeling like one must set aside time for learning & development. By using these casual

opportunities to learn, the student can create a lifelong habit of continuous learning.

Googling

Of course many students would straight-away google the error. This is appropriate in many situations, and often gets the student unstuck. However, it can lead to the copy paste trap where the student layers Stack Overflow answer upon answer, until their code works. Even if this leads to working code, it's likely a messy approach, and it's possible that the student doesn't fully understand the why of their solution. It's also likely that they cannot explain why this code is the most appropriate solution for their problem. A great standard to set for students is to set out the clear guidance that they must be able to explain the what and why of every line of code they submit.

Artificial Intelligence

Nowadays of course, it is more likely that a student will put their error or whole code into an AI chatbot. This can be a great option if handled appropriately, as the students can engage in conversation about the error, leading to deeper understanding. A curious and engaged student will enhance their learning through the conversation, but the pitfalls are obvious. A student who is looking to simply submit an assignment on time is likely to skim through the explanation and take the correct answer.

A great way for students to leverage AI is by using it to demystify some of the harder-to-parse sources, like documentation. The student can paste the part of the documentation that they believe relates to their issue and ask the AI to clarify or simplify the language.

As explained earlier as it relates to AI, the limitations of this technology should be made clear to students. It can, and indeed does, confidently hallucinate answers and provide references that do not exist. Students should verify the knowledge passed on by AI with external sources, especially if the AI's suggestions start to sound implausible or make the code worse.

Tip for Educators

Provide your students with an AI prompt that they can put into their chatbot which asks the agent not to give the students any code, and instead act as a Socratic Teacher who will engage in conversation about the topics and guide the student towards their understanding. For example:

```
I am a student learning Python. I am currently stuck on an error and I want to practice my debugging skills. Please act as a Socratic tutor. Do NOT give me the corrected code. Instead, ask me leading questions about my logic, point out specific lines where my mental model might be mismatched with the syntax, and help me arrive at the solution myself.
```

BUDDY: The Power of Collaboration

The most incredible thing about the buddy step, is that it doesn't require a buddy at all. Many programmers are extremely familiar with solving their own problem halfway through explaining it to a colleague! So, let's help students embrace the power of articulating their problem.

Rubber Ducking

A very common practice for software engineers is rubber ducking. This is when someone takes a rubber duck that is sitting on their desk for this express purpose, and one explains their code to them. In the process of performing this explanation, one usually sees the gap in the logic, or the misunderstood assumption, and has at least one new approach they can try to get themselves unstuck. While the rubber duck is a common talisman, this explanation can be done to anything: your cat, your favourite lego figure, or even out loud to yourself! My personal favourite that generates less public concern when you're among colleagues is to write down the explanation of the logic on paper as if I were speaking it out loud.

Artificial Intelligence

Of course, as mentioned in the section above, if a student chooses to use AI they should primarily use them as a Socratic tutor who they can have a discussion with. For me, this places AI firmly in the category of Buddy. Using the prompt above, the student should be able to engage with the AI as they would a colleague, with the great advantage of its 24/7 availability.

Fellow Humans

The two prior options described are great resources for the middle of the night bugs. However, if there is an option to discuss with an actual person, I would recommend this option above all.

Explaining your reasoning to someone else holds you to a higher standard of use of technical language. Whenever

you describe your code to someone else, you organise your thoughts before you speak and you take cues from them to expand on explaining the parts that are most important. Their intuition collaborates with yours in real time to create better understanding. You'd be surprised how often discussing something with a colleague will also enhance their understanding of concepts too.

Finally, learning is a collaborative endeavour: it becomes more effective and more enjoyable when shared.

Tip for Educators

Provide as many opportunities as you can to engage your students' technical collaboration skills. A great way to do this is by encouraging pair programming as part of tackling coding challenges.

Pair programming is an excellent practice for a myriad of reasons, (and rightly deserves a book in itself!), but for the scope of this chapter, its main advantage is that it develops the student's ability to explain their thinking using technical language. Students will collaborate in their problem solving and get each other unstuck during the pair programming session, leading to higher understanding and higher morale.

BOSS: When to Escalate

In my experience, students will err on the side of waiting too long to reach out, much to their detriment. A student's fear of seeming like they've not understood something they believe they should understand by now, or their pride in wanting to reach a solution by themselves, stops them from reaching out to their tutor. Time is a non-renewable learning resource and we must make this clear to our students.

Helping them build out the skill of reaching out to a superior at this early stage of their career will prove invaluable in a professional environment too, when not reaching out won't just be an impediment to their problem solving but possibly to their career advancement too.

A common first question is "how long do I wait?", and the reply is that it depends on the length of time you have to complete the task. If a student has been set a task in class that's 30 minutes long, and 10 minutes in they're still in the beginning stages, then it's appropriate to reach out. If they're working on an assignment that they have weeks to complete, they ought to try for longer than 10 minutes before reaching out.

Tip for Educators

The hardest part of reaching out can be the intimidation of knowing, ironically, that they've probably now waited longer than they should have. To encourage the students to reach out when appropriate and push through that feeling, it can be helpful to give them a framework of preparation they can use as a guide to feel more comfortable in escalating.

- **Context** - What are you trying to do? What part of what you're tacking is new to you? What specific error message or bug are you getting?
- **Brain** - What is your understanding of the problem? What hypotheses have you tested?
- **Book** - Where did you go to get more information? Was it helpful? Did it confuse you further? Do you have any similar sample code which you do understand?
- **Buddy** - Have you talked it through with a colleague? Did you discuss this problem with AI? Where in their explanations did you get lost?
- **Question** - What one specific question could your tutor answer you that would lead to the most significant unblocking?

Ensure you treat a student's escalation with utmost care, they are likely building a fragile muscle which will require tackling one's own ego and some old stubborn habits.

BREAK: The Essential Reset

Much like the ego that gets in the way of escalating, a similar feeling can get in the way of the key step of taking a break. Switching from the focused mode of active problem solving to diffuse mode, where problems are processed in the background. A 5 minute break to drink a glass of water or stare into the distance out the window is often much more productive than insisting on going through 5 more minutes of debugging.

The break step can, and arguably should, go in between any of the previous steps outlined. It should not be a last resource when the student has been at a problem uninterruptedly for an hour. Any time that the current step is making things less clear instead of more clear, Break should be utilised. As an instructor, I set out breaks during the lesson to model this approach.

It's also important to note that while short breaks should be part of their workflow, long breaks should be part of their life. It's our responsibility as educators to give our students a holistic perspective on learning, which incorporates time away from screens entirely. In the specific context of software bootcamps for example, students put a lot of pressure on themselves to continuously code and improve their skills. It's key to remind them that in order to learn well the next day, they will need some time in their day where they are doing something that refills their energy tank like eating well, exercising, and sleeping.

Conclusion

This framework codifies what is a natural, iterative loop for experienced problem solvers. By sharing the 5Bs mnemonic, we provide students with more than just a debugging checklist: we give them a toolkit for student resilience and a way to navigate debugging without falling into demotivation.

As educators, our role is to consistently remind them of the core principles that underpin this framework:

- **Errors are the Path to Progress** - We must move students away from seeing errors as failure and toward seeing them as the lane bumpers of their journey.
- **Time Management Practice** - We must emphasize that time is a non-renewable learning resource. Students should be taught that escalating to a Boss is an act of professional efficiency, not a sign of weakness.
- **The Power of the Reset** - We must model the Break as a technical requirement. Stepping away allows the brain to defocus, which is often when the most elegant solutions surface.
- **Thoughtful AI Integration** - By using AI as a buddy or a documentation translator, students leverage modern tools without sacrificing their journey of understanding

The 5Bs are a useful way to remember the common steps in problem solving, but the goal is learning to live in the loop.

References

Claxton, Guy. 2018. *The Learning Power Approach: Teaching Learners to Teach Themselves (the Learning Power Series)*. Crown House Publishing Ltd.