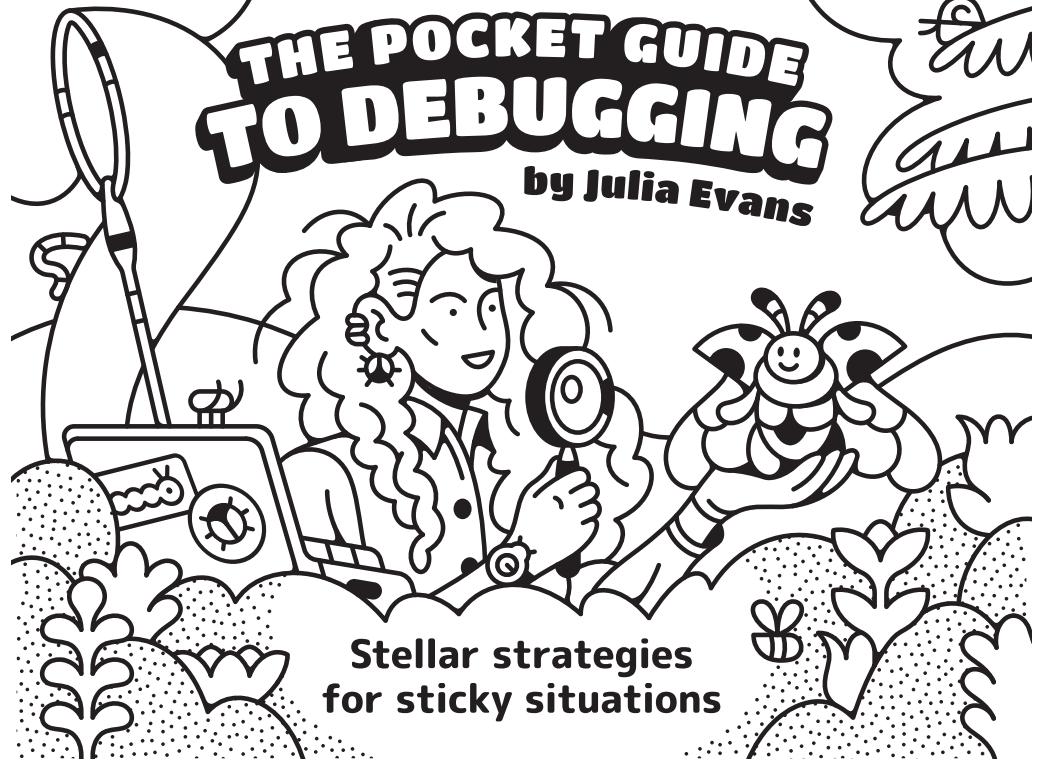
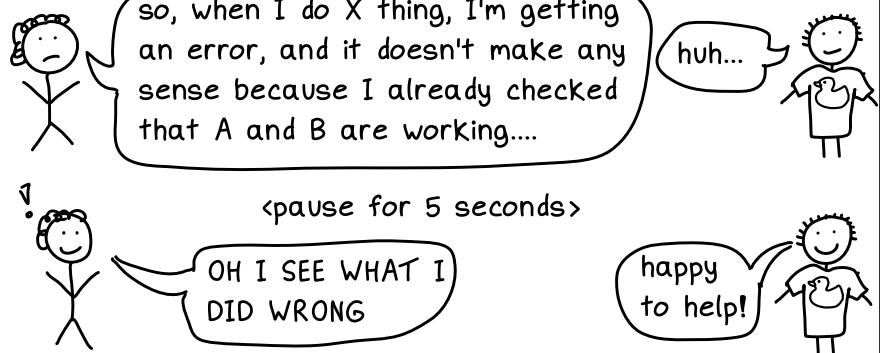


love this?
more at
★wizardzines.com★



explain the bug out loud

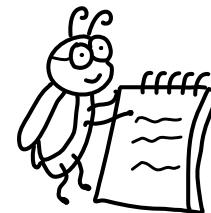
Explaining what's going wrong out loud is magic.



People call this "rubber ducking" because the other person might as well be a rubber duck (they don't say anything!)

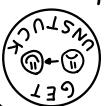
chapter 2

GET ORGANIZED



When I'M REALLY stuck, I'll write an email to a friend:
 write a message asking for help

- "I did X and I expected Y to happen, but instead..."
- "Here's what I'm trying to do..."
- "I've tried A, B, and C to fix it, but..."
- "This seems impossible because..."
- "Could this be because...?"
- "I did X and I expected Y to happen, but instead..."
- "This helps me organize my thoughts, and often by the time I finish writing, I've magically fixed the problem on my own!"
- "It has to be a specific person, so that the imaginary version of them in my mind will say useful things :)"



Cover art: Vladimir Kasićović
 Copy editing: Gerhardine La Fleche
 Editing: Dolly Lanuza, Kamal Marhubi
 Pairing: Marie Claire LeBlanc Flanagan
 and thanks to all the beta readers !!

credits

<https://mysteries.wizardzines.com>

One more thing: I also built a choose-your-own-adventure computer networking mysteries:
 debugging game to go with this zine, where you can solve
 computer networking mysteries:

thanks for reading

could I be using the wrong version of this library?
 sometimes I find it easier to think clearly when writing
 am I passing the wrong argument to function X?
 is something wrong with my hand on paper
 no filter! even the server?
 is the entire internet ridiculous?

Brainstorming every possible cause I can think of helps me not get stuck on the 1 or 2 most obvious possibilities.

brainstorm some aspects



document your quest



For very tricky bugs, writing up an explanation of what went wrong and how you figured it out is an amazing way to share knowledge and make sure you really understand it.

Ways I've done this in the past:

- ★ complain about it in the internal chat! ↗ so people can search for it!
- ★ write a quick explanation in the commit message
- ★ write a fun blog post telling my tale of woe!
- ★ for really important work bugs, write a 5-page document with graphs explaining all the weird stuff I learned along the way

62

about this zine

This zine has:



- ① a ``manifesto'' with my general debugging principles
- ② a list of my favourite debugging ``strategies'', which you can try in any order that makes sense to you



timebox your investigation



Sometimes I need to trick myself into getting started:



UGH, I do NOT want to look at this CSS bug!!!!

Giving myself a time limit really helps:



Okay, I'll just see what I can figure out in 20 minutes...



... 15 minutes later ... ↗



all fixed! that wasn't so hard!

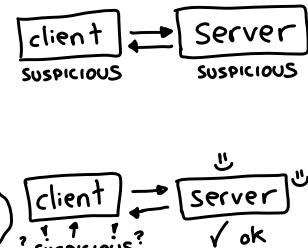
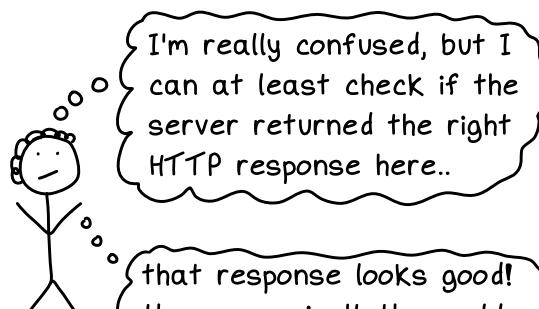
you can't always solve it in 15 minutes, but this works surprisingly often!

46

rule things out



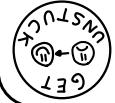
Once I have a list of suspects, I can think about how to eliminate them.



here we're assuming that was the only request being made. Otherwise this wouldn't be a safe conclusion :)

19

investigate the bug together



I find investigating a bug with someone else so much more fun than doing it alone. Debugging together lets you:

Adding a comment can help future you (or your coworkers!) avoid accidentally reviving a bug later.

 "I'll remember why I added this code, I spent 5 hours debugging it!"

Otherwise you would have written the code that may in the first place! You might think:

Some bug fixes are a little counterintuitive.

add a comment



keep a log book



I don't usually write things down. But 2 hours into debugging, I get really confused:

Wait, what did that error message say again exactly??

saw 2 hours ago say again exactly??

did I already try this???



Keeping a document with notes makes it WAY easier
stay on track. It might contain:

- specific inputs I tried
- error messages I saw
- stack overflow URLs

The log makes it easier to ask for help later if ne

② get organized

(1) first steps	manifester
(2) get organized	draw a diagram
(3) investigate	keep a log book
(4) research	rule things out
	brainstorm some suspects
add lots of print statements	find a new source of info
use a debugger	read the library's code
jump into a RPEL	learn one small thing
use a version that works	find the type of bug
look at recent changes	read the docs
sparkle assertions everywhere	analyze the logs
comment out code	write a failing test
identify one small question	rerun the code's steps
inspect unreproducible bugs	identify one small question
reproduce the bug	write a failing test
reread the error message	read the error message
preserve the crime scene	preserve the crime scene
read the error message	read the error message
rerun the code's steps	rerun the code's steps
jump into a RPEL	jump into a RPEL
use a debugger	use a debugger
add lots of print statements	add lots of print statements
	rule things out
	brainstorm some suspects
	keep a log book
	draw a diagram

find related bugs



When you're done fixing a bug, glance around to see if there are any obvious places in your code that have the same bug.

I was calling function X wrong, I'll check if we're calling that function wrong anywhere else!



wow, my assumption about how Y worked was TOTALLY wrong, I should go back and fix some things...

60

take a break ☕



Investigating a tricky bug requires a LOT of focus.

googling the same error message for the 7th time



ugh, nothing is working...

very frustrated

Instead, try one of these magical debugging techniques:
(even a 5 minute break can really help!)

get a coffee!

go to bed!

ride your bike!

eat lunch!

have a shower!



44

⑤ simplify ↗

- write a tiny program..... 38
- one thing at a time..... 39
- tidy up your code..... 40
- delete the buggy code..... 41
- reduce randomness..... 42

⑥ get unstuck ☹→☺

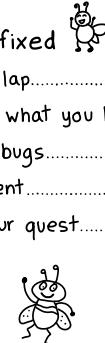
- take a break..... 44
- investigate the bug together..... 45
- timebox your investigation..... 46
- write a message asking for help... 47
- explain the bug out loud..... 48
- make sure your code is running..... 49
- do the annoying thing..... 50

⑦ improve your toolkit

- try out a new tool..... 52
- types of debugging tools..... 53
- shorten your feedback loop..... 54
- add pretty printing..... 55
- colours, graphs, and sounds..... 56

⑧ after it's fixed 🎊

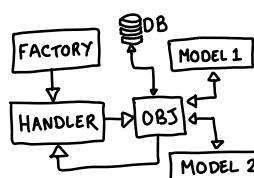
- do a victory lap..... 58
- tell a friend what you learned..... 59
- find related bugs..... 60
- add a comment..... 61
- document your quest..... 62



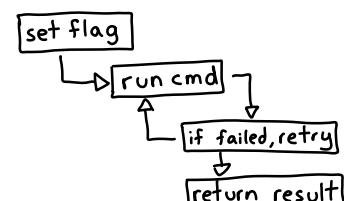
draw a diagram

Some ideas:

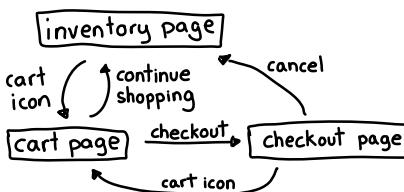
network diagram



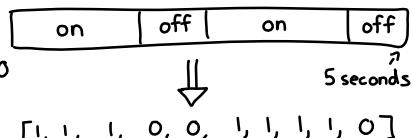
flowchart



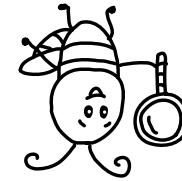
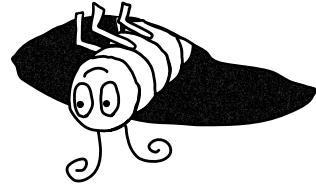
state diagram



or anything else
(like a data structure!)



21



GET UNSTUCK

chapter 6

I love to celebrate squashing a bug by telling a friend:
tell a friend what you learned

Some possible outcomes of this:

- ↳ they've seen that bug too, and teach me something else!
- ↳ they ask questions I hadn't thought of
- ↳ they tell me about a website/tool I didn't know about
- ↳ it helps solidify my knowledge!

59

INVESTIGATE

chapter 3

1 inspect, don't squash

2 being stuck is temporary

3 trust nobody and nothing

4 it's probably your code

5 slowly growing horror

6



a debugging manifesto

do a victory lap



Once you've solved it, don't forget to celebrate! Take a break! Feel smart!



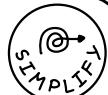
now is not the time for humility

The best part of understanding a bug is that it makes it SO MUCH easier for you to solve similar future bugs.



58

reduce randomness



It's much easier to debug when your program does the exact same thing every time you run it.



the bug only happens 10% of the time, it's SO HARD to figure out if my change fixed it or not

There are a bunch of tools for controlling your program's inputs to reduce randomness, for example:

- many random number generators let you set the seed so you get the same results every time
- faketime fakes the current time
- libraries like Ruby's vcr can record HTTP requests
- record/replay debuggers like rr record everything

42



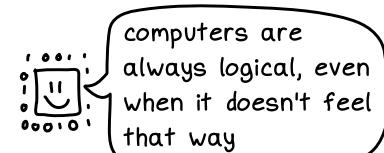
5 don't go it alone



WHAT IS HAPPENING?!?

what if we try X?

6 there's always a reason



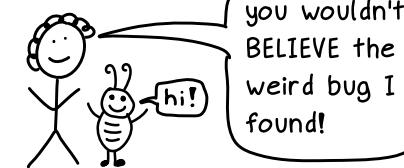
computers are always logical, even when it doesn't feel that way

7 build your toolkit



wow, the CSS inspector makes debugging SO MUCH EASIER

8 it can be an adventure



add lots of print statements

I love to add print statements that print out 1, 2, 3, 4, 5...



console.log(1)
console.log(2)
console.log(3)

using descriptive strings is smarter, but I usually use numbers or "wtf???"

This helps me construct a timeline of which parts of my code ran and in what order:

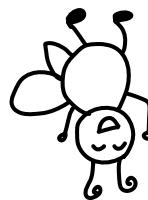


Often I'll discover something surprising, like "wait, 3, never got printed??? Why not???"

23

AFTER IT'S FIXED

chapter 8



delete the buggy code

Sometimes the buggy code is not worth salvaging and should be deleted entirely. Reasons you might do this:

• this library isn't working, I'm going to switch to `y instead`

- it uses a confusing library / tool

I bet I could avoid all these problems if I took X approach instead...

- idea for how to implement it

I just have to reproduce the bug once they make hard-to-reproduce bugs easier:

I love record/replay debuggers because programs execution and time travel! "record replay" debuggers let you record your entire program any time it's modified * watch a location in memory and stop * jump into a REPL to poke around (see page 25)

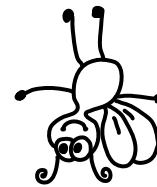
Your debugger might let you:

Some languages' debuggers have more features than others.

A debugger is a tool for stepping through your code line by line and looking at variables. But not all debuggers are equal!

Line and column numbers are useful for navigating through your code.

use a debugger



FIRST STEPS

chapter 1

41

42

colours, graphs, and sounds

Instead of printing text, your program can tell you about its state by generating a picture! Or playing sounds at key moments!

Some ways your programs can generate pictures or sounds:

- ★ add colours to your log lines
- ★ add red outlines around every HTML element!
- ★ Haskell has an option to beep (🔔) at the start of every major garbage collection
- ★ draw a chart of events over time
- ★ use graphviz to generate a diagram of your program's internal state



56

preserve the crime scene

One of the easiest ways to start is to save a copy of the buggy code and its inputs/outputs:



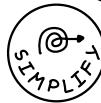
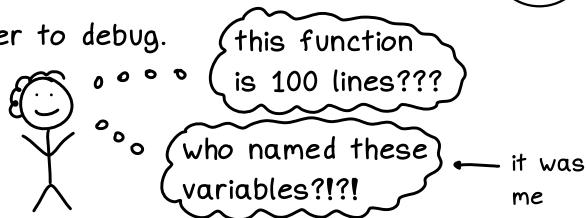
Depending on the situation, you might want to:

- make a git commit of the buggy code!
(on a branch, just for you)
- save the input that triggered the bug
- save logs/screenshots to analyze later

q

tidy up your code

Messy code is harder to debug.



Doing a tiny bit of refactoring can make things easier, like:

- rename variables or functions
- format it with a code formatter (go fmt, black, etc.)
- add comments
- delete old/untrue comments

Don't go overboard with the refactoring though: making too many changes can easily introduce new bugs.

40

jump into a REPL

In dynamic languages (like Python / Ruby / JS), you can use a debugger to jump into an interactive console (aka "REPL") at any point in your code. Here's how to do it in Python 3:

① edit your code `my_var = call_some_function()`
`breakpoint()` ← add this!

② refresh the page

③ play around in the REPL! You can call any function you want / try out fixes!

How to do it in other languages:

- ★ Ruby: binding.pry
- ★ Python (before 3.7): import pdb; pdb.set_trace()
- ★ Javascript: debugger;



25

39

one thing at a time

It's tempting to try lots of fixes at once to save time:

reality:

... now there's a new problem AND it's still broken

dream:

I'm going to add Z, and replace X with Y, and improve C - that'll definitely fix it!

- If I found I've done this by accident, I'll:
- Undo all my changes (git stash!)
- make a list of things to investigate, one at a time

26

find a version that works

If I have a bug with how I'm using a library, I like to:

- find a code example in the documentation
- make sure it works
- slowly change it to be more like my broken code
- test if it's still working after every single tiny change
- know that DESEN'T cause the bug to come back, I know that change wasn't the problem.

55

add pretty printing

Sometimes you print out an object, and it just prints the class name and reference ID, like this:

```
MyObject<#18238120323>
```

Implementing a custom string representation for a class you're often printing out can save a LOT of time.

The name of the method you need to implement is:

```
__str__
```

Python: .`__str__` Ruby: .`to_s` JavaScript: .`toString`

Java: .`toString` Go: `String`

Also, pretty-printing libraries (like pprint in Python or awesome-pprint in Ruby) are great for printing out arrays/hashmaps.

10

read the error message

Error messages are a goldmine of information, but they can be very annoying to read:

giant stack trace full of irrelevant to your bug

often seems totally unrelated to your bug

impenetrable jargon, like "misleading, like permission denied"

can even be "doesn't exist"

unrelated to your bug

tricks to extract information from giant errors: If there are many different error messages, start with the first one. Fixing it will often fix the rest.

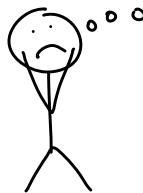
If the end of a long error message isn't helpful, try looking at the beginning (scroll up!)

* On the command line, pipe it to less so that you can scroll/search it (`./my-program >8 | less`)

* If you don't include `>8`, less won't show you the error messages (just the output)

shorten your feedback loop

When you're investigating a bug, you'll need to run the buggy code a million times.



UGH, I need to type all this information into the form to trigger the bug AGAIN??? This is literally the 30th time :(:(

Ways to speed it up:

- * use a browser automation tool to fill in forms / click buttons for you!
- * write a unit test!
- * autorun your code every time you save!



54

reread the error message

After I've read the error message, I sometimes run into one of these 3 problems:

- ① misreading the message



ok, it says the error is in file X

spoiler: it actually said file Y

- ② disregarding what the message is saying



well, the message says X, but that's impossible...

spoiler: it was possible

- ③ not actually reading it



ok, I read it...

spoiler: she did not read it

11

write a tiny program



Does your bug involve a library you don't understand?



UGH, requests is NOT working how I expected it to!

I like to convert my code using that library into a tiny standalone program which has the same bug:



→ ≈
20 lines of
buggy code

I find this makes it WAY EASIER to experiment and ask for help. And if it turns out that library actually has a bug, you can use your tiny program to report it.

38



look at recent changes

Often when something is broken, it's because of a recent change. Usually I look at recent changes manually, but git bisect is an amazing tool for finding exactly which git commit caused the problem.

We don't have space for a full git bisect tutorial here, but here's how you start using it:

```
git bisect start  
git bisect bad HEAD  
git bisect good 1fe9dc
```

ID of a commit that
doesn't have the bug

Then you can either tag buggy commits manually or run a script that does it automatically.

27

try out a new tool



There are TONS of great debugging tools (listed on the next page!), but often they have a steep learning curve. Some tips to get started:

- get someone more experienced to show you an example of how they'd use the tool this is SO helpful!!!
- try it out when investigating a low stakes bug, so it's no big deal if it doesn't work out
- take notes with examples of the options you used, so you can refer to them next time

52

find a new source of info



We all know to look at the official documentation. Here are some less obvious places to look for answers:

- * the project's Discord, Slack, IRC channel, or mailing list
- * code search (search all of GitHub for how other people are using that library!)
- * GitHub issues (did someone else have the same problem?)
- * release notes (is the bug fixed in the new version?)
- * a book chapter (you might have a book on this topic!)
- * blog posts (sometimes there's an amazing explanation on the 2nd page of Google results)

36

inspect unreproducible bugs



When you can't reproduce a bug locally, it's tempting to just try random fixes and pray. Resist the temptation! Some ways to get information:

- try to reproduce the environment where it happened
- ask for screenshots / screen recordings
- add more logging, deploy your code, and repeat until you understand what caused the bug
- read the code VERY VERY carefully incredibly boring but it actually does work sometimes
- do your experimentation somewhere where you *can* reproduce the bug on a staging server? on someone else's computer?

13

comment out code



Commenting out code is an amazing way to quickly do experiments and figure out which part of your code is to blame. You can:

- * comment out a function call and replace it with a hardcoded value, to check if the function call is broken
- * if the error message doesn't give you a line number, comment out huge chunks of the program until the problem goes away
- * comment out some code and rewrite it to see if the new version is better

29

do the annoying thing

Sometimes when I'm debugging, there are things I'll refuse to try because they take too long.



But as I become more and more desperate, eventually I'll give in and do the annoying thing. Often it helps!



50



retrace the code's steps

Here's a classic (but still very effective!) way to get started:

- ① find the line of code where the error happened
- ② trace backwards to investigate what could have caused that error. Keep asking "why?"

There's an error on line 58...

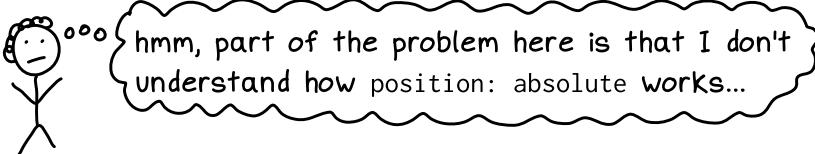
- ↳ that's because this variable has the wrong value...
- ↳ the value is set by calling this function...
- ↳ that function is making an HTTP request to the API...
- ↳ the API response doesn't have the format I expected! Why is that?



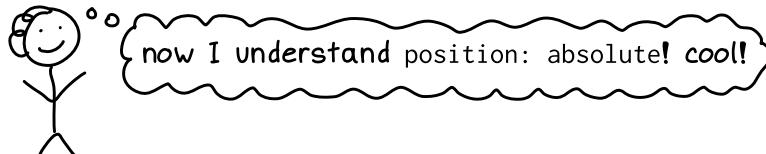
15

learn one small thing

Bugs are a GREAT way to discover things on the edge of your knowledge.



Finding one small thing I don't understand and learning it is really useful (and pretty fun!)



34



chapter 4

RESEARCH



