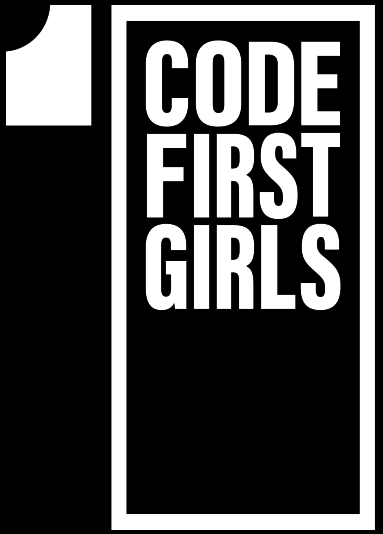
TROUBLESHOOTING WORKSHOP

LESSON 8



NANODEGREE → ENGINEERING MODULE

AGENDA

01 Debugging vs Troubleshooting

02 Debugging Techniques

03 Troubleshooting Techniques

04 Troubleshooting Activity

06 Engineering Interview Brain Teasers

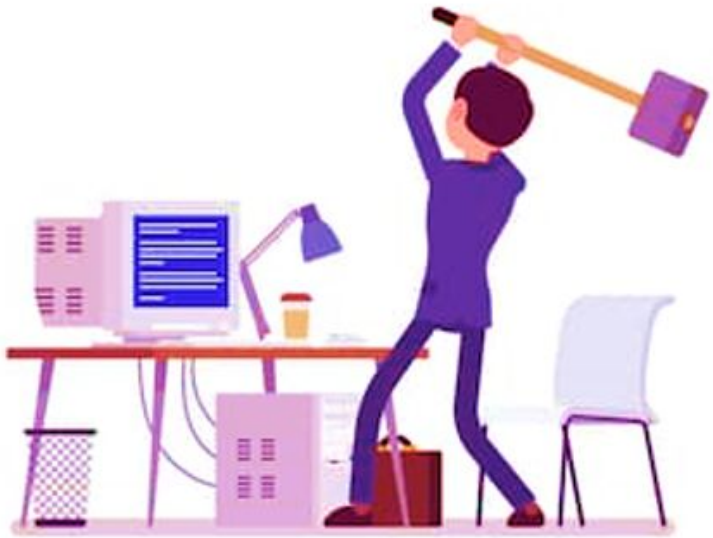
TROUBLESHOOTING vs DEBUGGING

- Troubleshooting and debugging are two concepts programmers need to know and distinguish between.
- We must understand the two terms and know how the common traits they share differ.

- **NOTE: your will spend more time with debugging than with troubleshooting in a day to day work.**
- **HOWEVER: your extended operational duties will cause you to troubleshoot more frequently.**

TROUBLESHOOTING

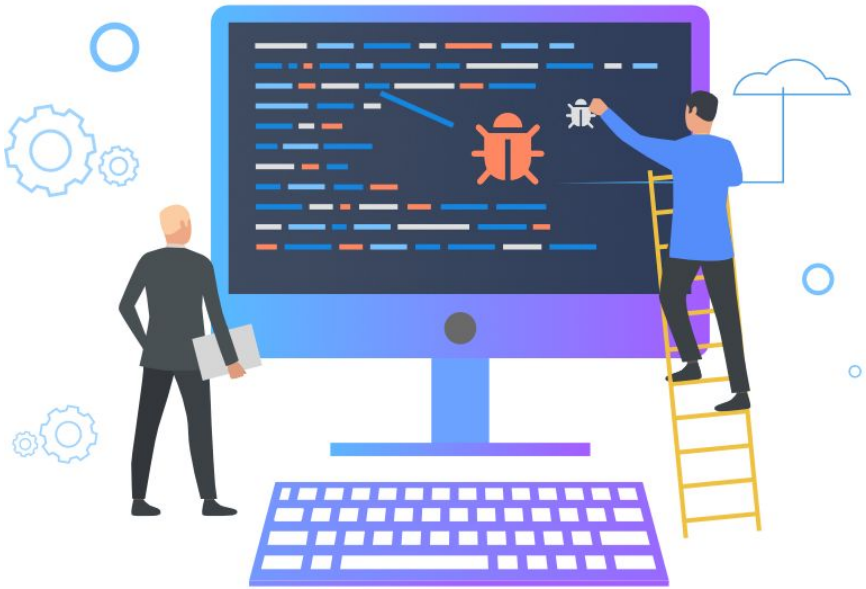
INTRODUCTION



- Troubleshooting is a process that helps people identify issues or problems occurring in a system.
- Troubleshooting tends to exist at a higher level than debugging and applies to many components of a system.
- It’s a process of parsing out the items that are causing problems.
- This process requires interviewing the end users of the system to find out the steps they took to cause the problems.

DEBUGGING

INTRODUCTION



- Debugging is a subset of troubleshooting.
- Debugging implies finding problems as they relate to computer code.
- When you are tasked with debugging a module of code, you find what is causing the problem and then you must fix it (NB: this is an oversimplification).
- There may be several points of failure in the code, and sometimes it's not obvious where the problems are occurring, so we must strive to break down the code into chunks and identify issues.

GOLDEN RULES

WHEN A PROBLEM STRIKES

Where do we even begin?

1. Know and apply debugging techniques
2. FInd what has changed
3. Use tools, but don't fully depend on them
4. Give everyone else the benefit of the doubt when debugging
5. Help others find the problem

DEBUGGING TECHNIQUES

OVERVIEW

“No programmer will ever write bug free code, but with practice and determination, it is possible to write clean code, keep bugs in their place and deliver reliable software systems.”

PRINT STATEMENT

DEBUGGING

print( )

1. Print statements are the fastest, easiest and most direct way to inspect the data values and types of variables.
2. Well-placed print statements allow us to track the flow of data through a piece of code and quickly identify the source of the bug.
3. The humble print statement is always a first tool we turn to when trying to debug a piece of code.

DEBUGGER

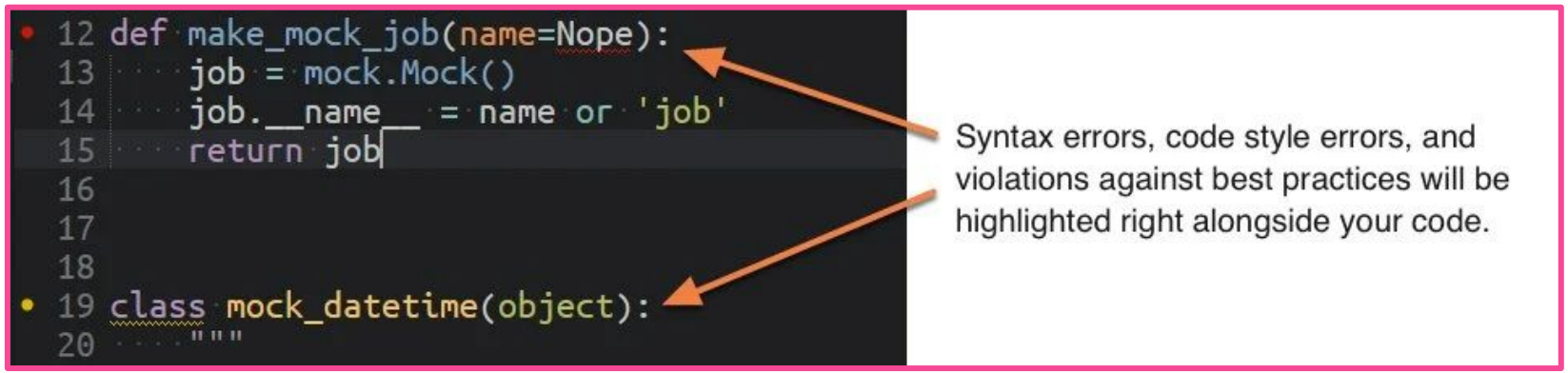
DEBUGGING

Pdb Library

1. Source code debuggers carry the print statement method of debugging to its logical conclusion.
2. They allow a programmer to step through code execution line by line and inspect everything from the value of variables to the state of the underlying virtual machine.
3. Employing a debugger can be overkill, but when it is used properly, a debugger can be a powerful and efficient tool

LINTER

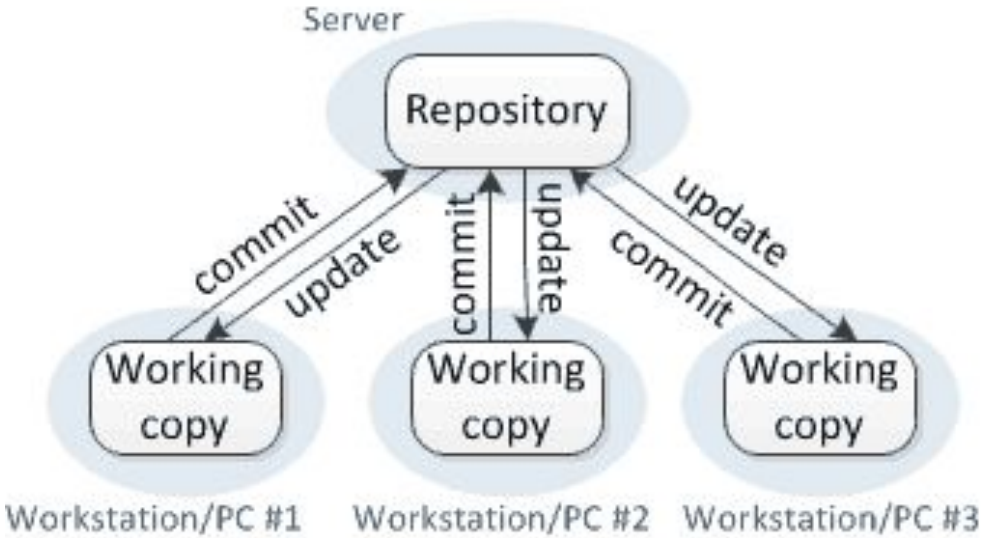
DEBUGGING



1. Linter can perform static analysis on code to recognize problem areas before the code is compiled or run, and in some languages a lint tool is useful for syntax checking and style enforcement.
2. Running a lint program inside of an editor while writing code, or passing code through a linter before compiling or running the code helps programmers find and correct errors before they arise as bugs in the executed software.

VERSION CONTROL

DEBUGGING



Managing code versions between branches and the master build
EXAMPLE: Git and GitHub

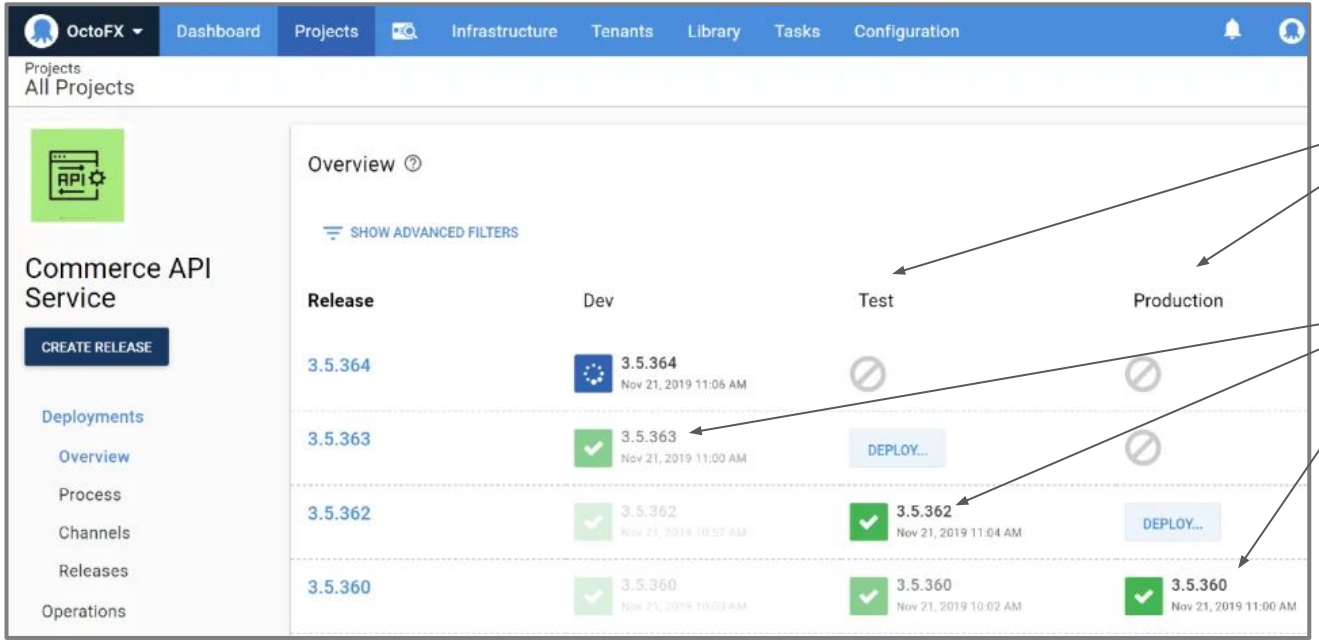
1. Using a version control system is a software engineering best practice that we cannot afford to ignore.

2. The different versions can then be merged together, so multiple programmers can work on a code base simultaneously.

3. Version control systems are also crucial because they give programmers the ability to rollback changes to an earlier version of the code!

VERSION CONTROL

DEBUGGING



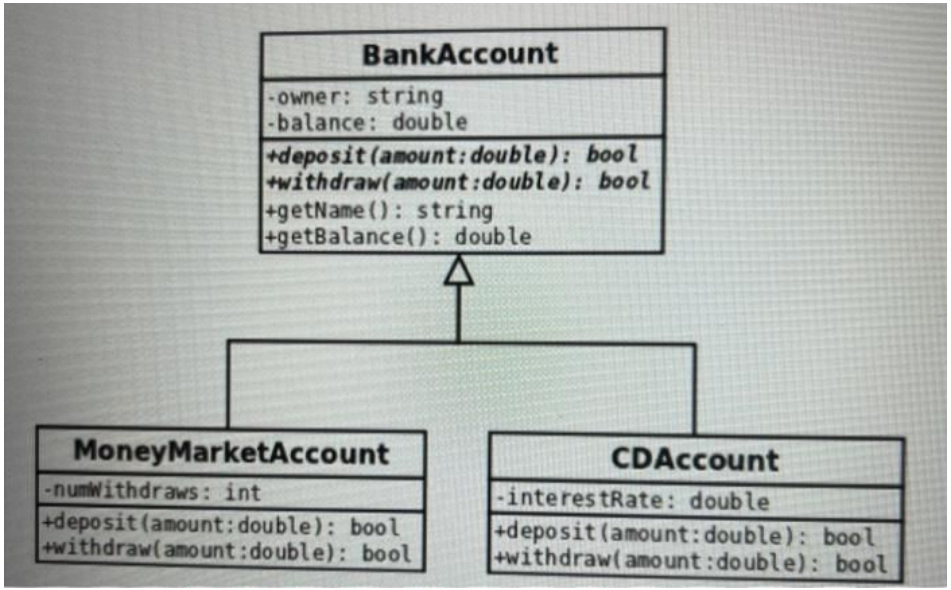
environments

master versions

Managing master versions between different environments
EXAMPLE: Octopus Tool

MODULARIZATION

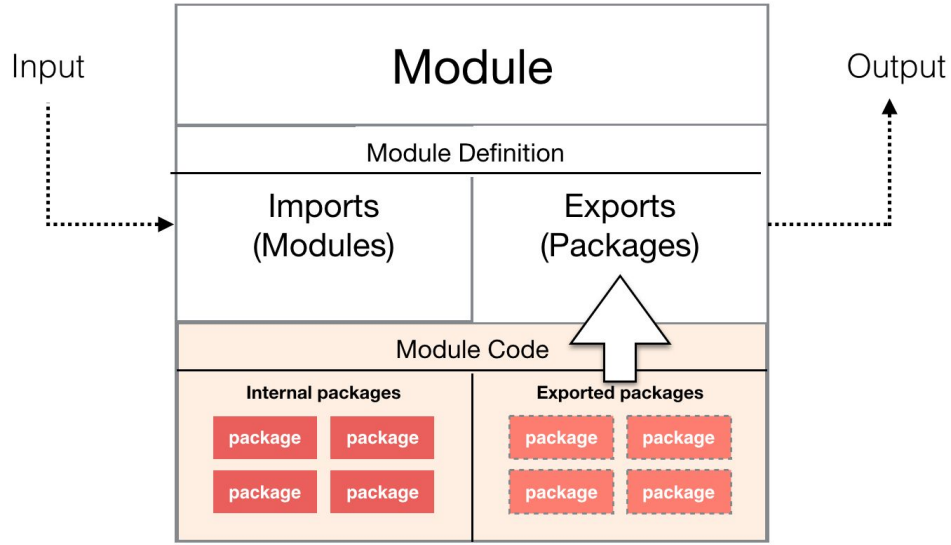
DEBUGGING



1. Poorly architected code is a major source of hard-to-fix bugs.
2. When code is easy to understand and can be "executed" mentally or on paper, there is a good chance that programmers can find and fix bugs quickly.
3. The best way to ensure this is to write functions/classes that only do one thing.
4. Remember, a piece of code with many responsibilities has many opportunities for errors that are difficult to track down.

MODULARIZATION

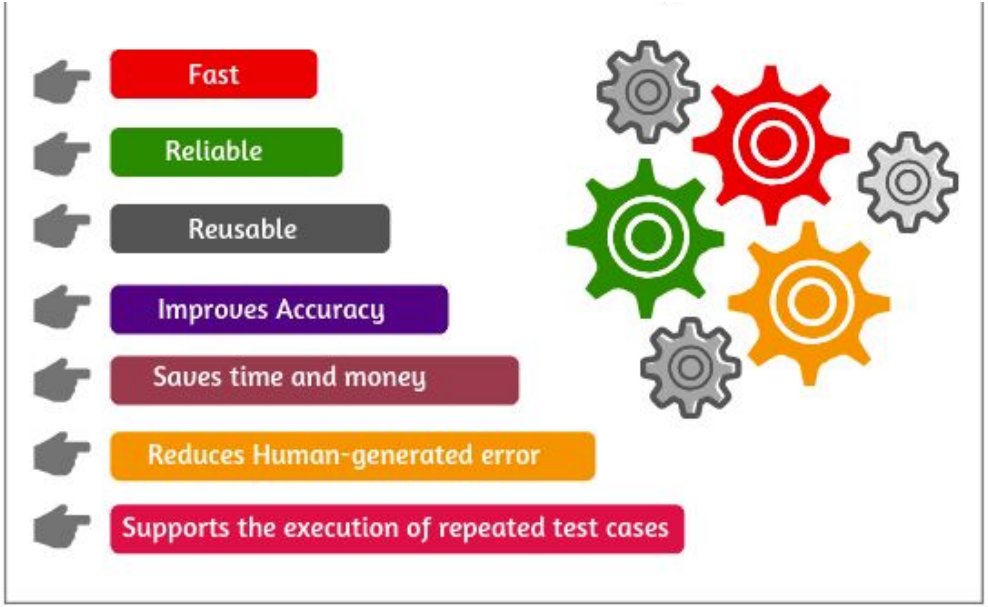
DEBUGGING



1. Designing software components that handle only one concern is often called code modularization.
2. Modularization helps programmers understand software systems in two ways:
   a. First, modularization creates a level of abstraction that makes it possible to think of a module of the system without understanding all of the details
   b. Second, the details of a module can be examined and understood without being obfuscated by unrelated code.

AUTOMATED TESTS

DEBUGGING



1. Unit tests and other types of automated tests go hand in hand with modularization.
2. An automated test is a piece of code that executes software with specific inputs and checks to see if the program behavior matches what is expected.
3. The Python standard library includes a Python version called “PyUnit” or simply “unittest”

TEDDY BEAR METHOD / RUBBER DUCK DEBUGGING

DEBUGGING

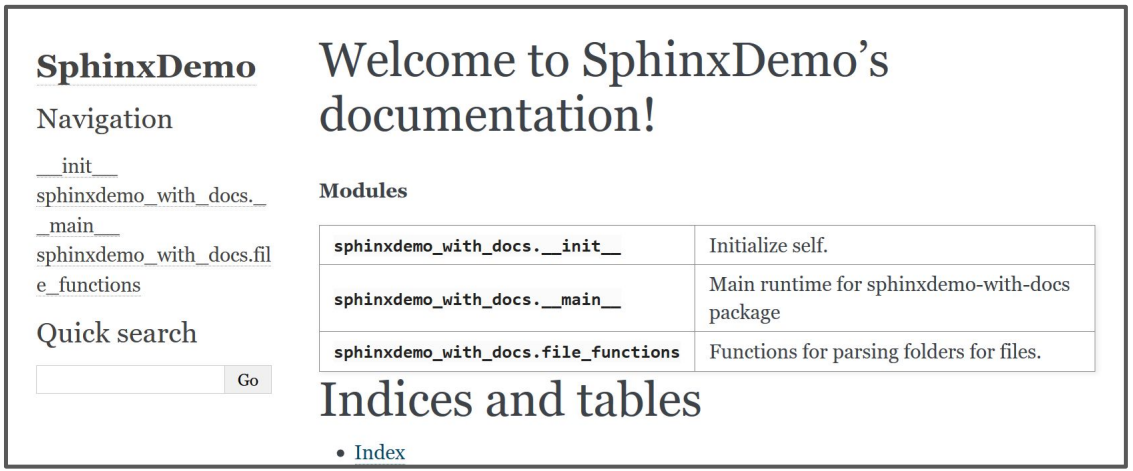
1. Rubber duck debugging originated in a university computer center where students were required to sit down across from a teddy bear and explain their bugs to it before they could seek help from a live person.
2. This method is so effective that it spread quickly throughout the entire software engineering world, and it persists to this day!
3. Nearly anything can be substituted for the teddy bear: rubber ducks are a popular choice, as are patient non-programmers.

COMMENTS and DOCUMENTATION

DEBUGGING

```python
# This class provides utility functions to work with Strings
# 1. reverse(s): returns the reverse of the input string
# 2. print(s): prints the string representation of the input object
class StringUtils:

    def reverse(s):
        return ''.join(reversed(s))
```



SphinxDemo

Navigation

__init__
sphinxdemo_with_docs._
__main__
sphinxdemo_with_docs.file_functions

Quick search

Go

Welcome to SphinxDemo's documentation!

Modules

`sphinxdemo_with_docs.__init__`	Initialize self.
`sphinxdemo_with_docs.__main__`	Main runtime for sphinxdemo-with-docs package
`sphinxdemo_with_docs.file_functions`	Functions for parsing folders for files.

Indices and tables

- Index

1. Comments should explain the purpose of code on a low-level.
2. Try writing readable code that is implemented as simply as possible and uses sensible names for functions and variables.
3. Software documentation describes the functionality of a software system.
4. Writing documentation demonstrates your understanding of the software system, and often points out the parts of the system that are not well understood and are a likely source of bugs.

STEPS TO TROUBLESHOOTING

OVERVIEW

A skill all software developers are expected to have:
the problem-solving process.

Nobody formally teaches this skill and often we are just supposed to “know how”.
Let’s review the key steps we can take to solve a problem.

KEY STEPS

TROUBLESHOOTING

1. Identify the problem
2. Gather information
3. Iterate potential solutions
4. Test your solution

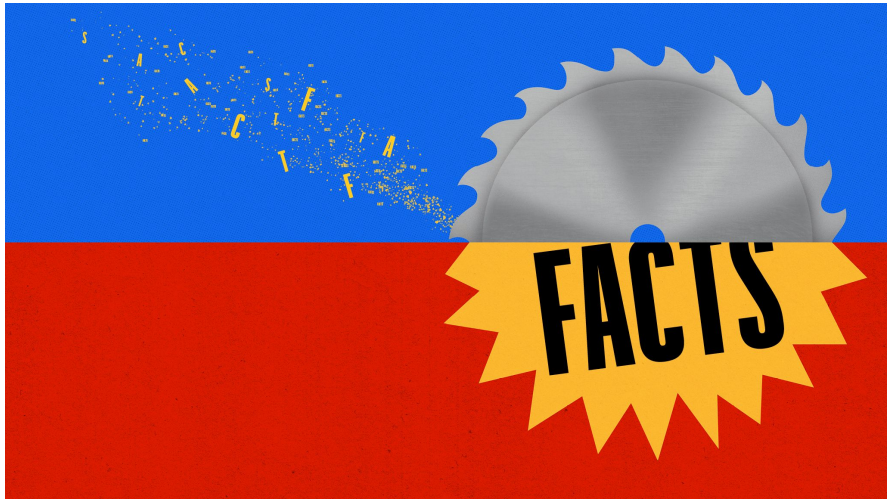
WHAT EXACTLY IS THE PROBLEM?

TROUBLESHOOTING

1. Take as many notes as possible.
2. Ask more investigative questions about the problem.
3. Ask for example:
    - what error message they get
    - payload they are using for an API
    - Data / values they are trying to input or process
4. Repeat back your understanding of the problem.
5. Ensure that you and the reporter are on the same page.

GATHER MORE DETAILS

TROUBLESHOOTING



1. Error messages
2. Events logs
3. Screenshot examples
4. Diagnostics
5. Monitoring tools

REPRODUCE THE PROBLEM / DEVELOP HYPOTHESIS

TROUBLESHOOTING

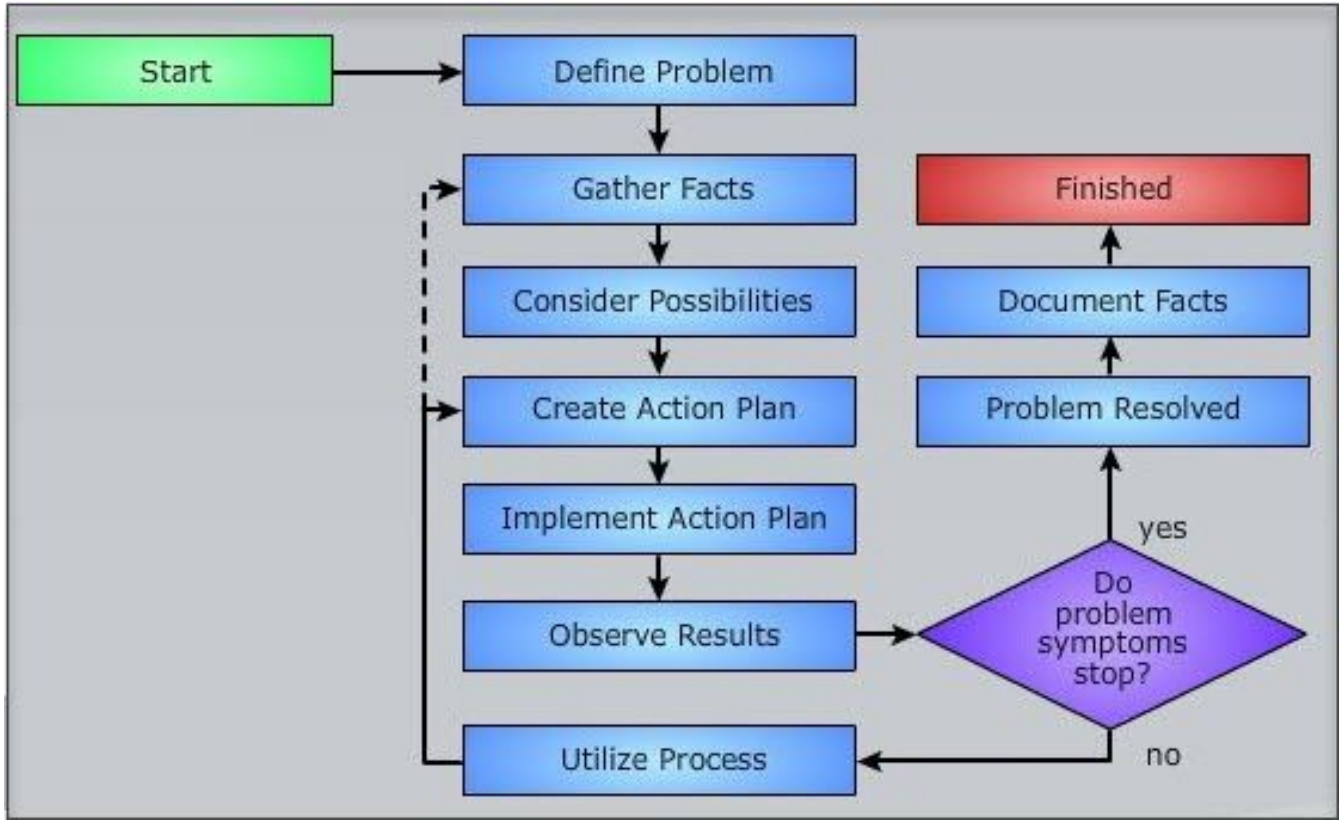
1. Try **reproducing the same issue in the non-Prod environment** (Dev / UAT).
2. You may use mock input to simulate the problem.
3. Once recreated, then try to identify and fix the root cause.

TEST YOUR SOLUTION

TROUBLESHOOTING

1. When a problem is solved, test your solution to ensure that issues are fixed and no new issues are introduced!
2. Engage your user in testing to make sure that now they are able to get expected results.

TROUBLESHOOTING



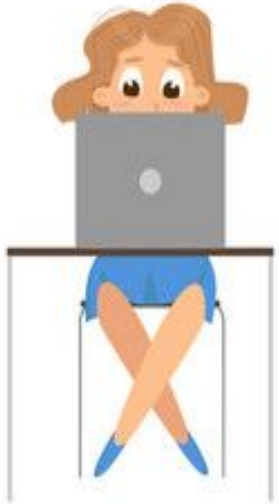
TROUBLESHOOTING SCENARIO

SETUP

1. We are a fun **international online store selling SOCKS worldwide**.
2. A customer can place an order online and we will ship it after 2 days from the moment an order is received.
3. We have just decided to add a new feature to our shopping system:
   a. **A customer can make an adjustment to his/her order, for example**: add more items, replace items or cancel the order.
   b. **The adjustment can only be made within the strict 2 days (48h) period** from the moment their order was placed.
   c. After 48h the order is NOT amendable and NOT refundable.

TROUBLESHOOTING SCENARIO

ROLES



DEV TEAM WHO WILL DO TROUBLESHOOTING

Can we have 3 volunteers?

TROUBLESHOOTING SCENARIO

ROLES

PROJECT MANAGER - YOUR INSTRUCTOR

- **PM can answer your questions**
- **PM has access to customers' data and knows their issues**
- **PM needs you to solve the issue ASAP!**

TROUBLESHOOTING SCENARIO

NEW FEATURE

SHOPPING SYSTEM VERSIONS IN DIFFERENT ENVIRONMENTS			
	Dev	UAT	PROD
Current verion	3.21.23	3.21.23	3.20.14
Previous versions	3.20.14	3.20.14	3.19.10
	3.20.13-pr68		
	3.19.10	3.19.10	
	3.19.45-pr10		

3.20.14 - the version which contains new feature (ability to post adjustments) was released last night

3.19.10 - previous version; it does not contain the new adjustment feature. We have been using this version successfully for the last 4 month, before we released a new one.

NOTES:

- PROD - this is the version of the shopping system that our clients use and see. They go to our website and place socks orders with our shop.
- 3.20.14 - before we released this version, it was thoroughly tested by the dev team and in-house business analysts. It was approved to be released.

TROUBLESHOOTING SCENARIO

PROBLEM

- We have just received a complaint from a customer saying that they tried to adjust their order, but the request could not be processed!
- They are unable to adjust the order at all.
- Note that for some customers it works OK and they can make adjustments.
- We need to do something about this issue ASAP.
- There are more customers who call to say that they cannot post adjustments.

TROUBLESHOOTING

INVESTIGATION

DEV TEAM NEEDS TO DEAL WITH THIS ISSUE, INVESTIGATE AND FIX THE PROBLEM.

- You can advise on immediate course of action
- You can ask your PM questions to get to the bottom of this.

If our Dev team is stuck at any point, then we can ask the rest of the group for their suggestions or leading questions.

BRAIN TEASERS FOR ENGINEERS

ACTIVITY FOR ALL

- A lot of tech companies and other employers seem to have a tendency for asking tricky questions to potential candidates to assess how they think.
- In some cases, there is no right answer per se, it's rather a way of assessing how your brain is wired.
- Let’s practice solving some of the Teaser Challenges from the top players like Facebook, Google, Goldman Sachs.
- There are plenty of examples, but we will practice only a few to “get the flavour”.

MOTORBIKES

BRAIN TEASER

- There are 50 bikes, each with a tank that holds enough gas to go 100 km.
- Using these 50 bikes, what is the maximum distance that you can go?

WATER

BRAIN TEASER

- If you had an infinite supply of water and a 5-liter and 3-liter bucket, how would you measure exactly 4 liters?

- The buckets do not have any intermediate markings.

SOCKS

BRAIN TEASER

- There are 20 different socks of two types in a drawer in a completely dark room.
- What is the minimum number of socks you should grab to ensure you have a matching pair?

BRAIN TEASERS FOR ENGINEERS

ACTIVITY FOR ALL

- Hopefully you enjoyed these BRAIN TEASERS
- There might be similar ones on your Mock Interview questions ;)

THANK YOU!