

Debugging like a Pro

GONZALO OSCO HERNANDEZ

**When
you fix in
Production**



What is debugging?

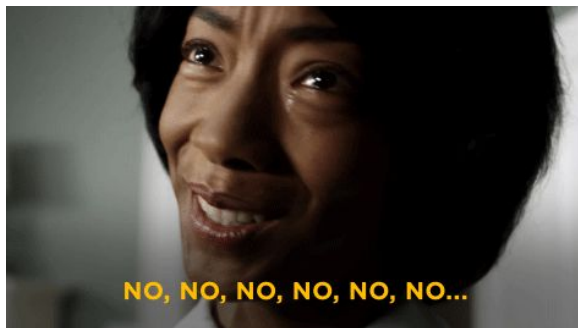


- Is “finding a fix”?... NO!
- Debugging is the process of finding and resolving **defects** or problems within a computer program that prevent correct operation of computer software or system.

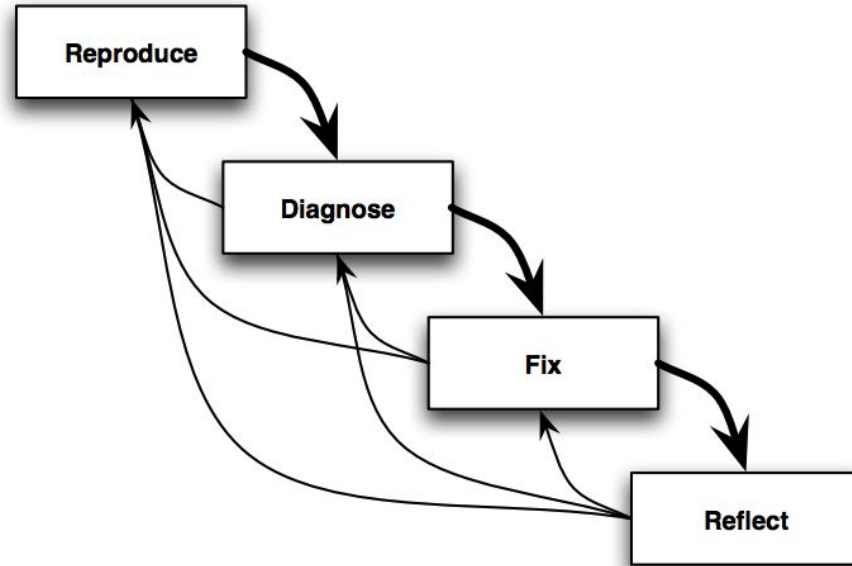
Effective Debugging (Like a dream)

Effective debugging requires that we take these steps:

1. **Work out** why the software is behaving unexpectedly.
2. **Fix** the problem.
3. **Avoid breaking** anything else.
4. **Maintain or improve** the overall quality of the code.
5. Ensure that **the same problem** does not occur elsewhere and **cannot occur again**.



The Core debugging Process



The Core Debugging Process

You need to know exactly

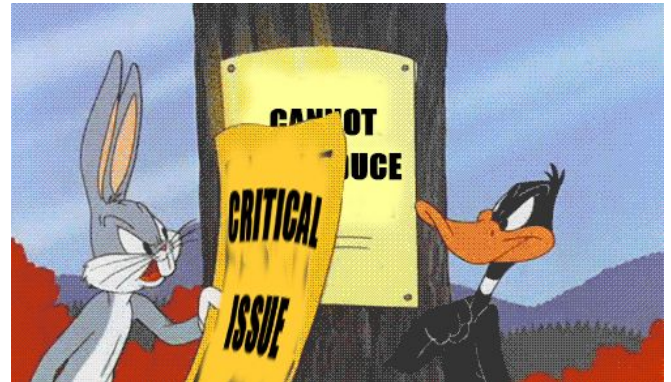
What is happening, and what should?

**Understanding is
Everything!**

Reproduce it!

Find a way to reliably and conveniently reproduce the problem on demand.

- Reproduce First, Ask Questions Later
- Start with the Obvious
- Controlling the **Software, Environment, and Inputs**
- Force Error Conditions
- Check not only the “**happy path**”
- Logging
- As Simple as Possible

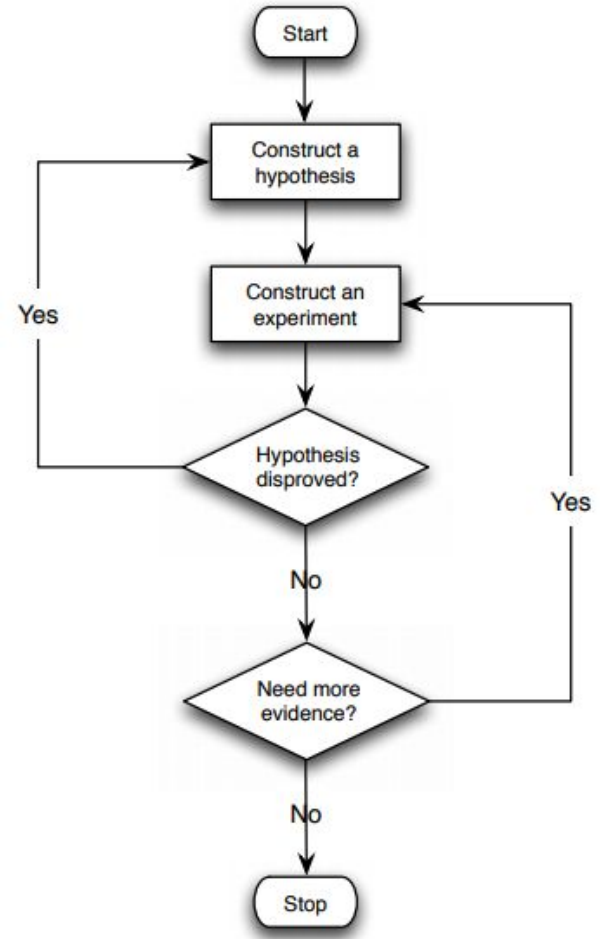
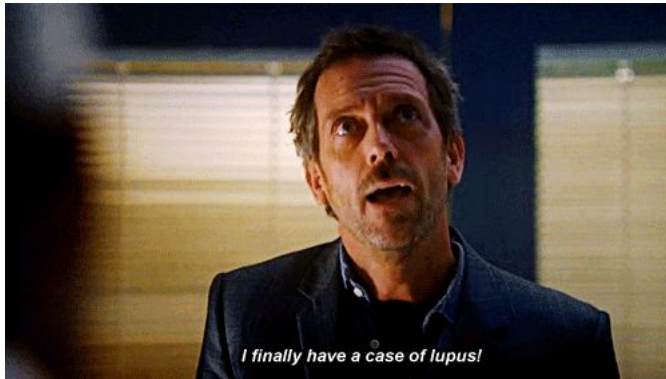


Reproduce it! - PUT IN ACTION

- ★ Find a reproduction before doing anything else.
- ★ Ensure that you're running the same version as the bug was reported against.
- ★ Duplicate the environment that the bug was reported in.
- ★ Determine the input necessary to reproduce the bug by:
 - – Inference
 - – Recording appropriate inputs via logging
- ★ Ensure that your reproduction is both reliable and convenient through iterative refinement:
 - – Reduce the number of steps, amount of data, or time required.
 - – Remove nondeterminism.
 - – Automate.

Diagnose it!

Construct **hypotheses**, and test them by performing **experiments** until you are confident that you have **identified the underlying cause of the bug**.



Diagnose it! - THE EUREKA MOMENT



I already know why it does not work!

Diagnose it! - PUT IN ACTION

- ★ Construct hypotheses, and test them with experiments.
 - Make sure you understand what your experiments are going to tell you.
 - Make **only one change at a time**.
 - Keep a record of what you've tried.
 - **Ignore nothing.** •
- ★ When things aren't going well:
 - If the changes you're making don't seem to be having an effect, you're not changing what you think you are.
 - **Validate your assumptions.**
 - Are you facing multiple interacting causes or a changing underlying system?
- ★ **Validate your diagnosis**

Fix it!

Design and implement changes that fix the problem, avoid introducing regressions, and maintain or improve the overall quality of the software.

Fix the Cause, NOT the Symptoms!



Fix it!

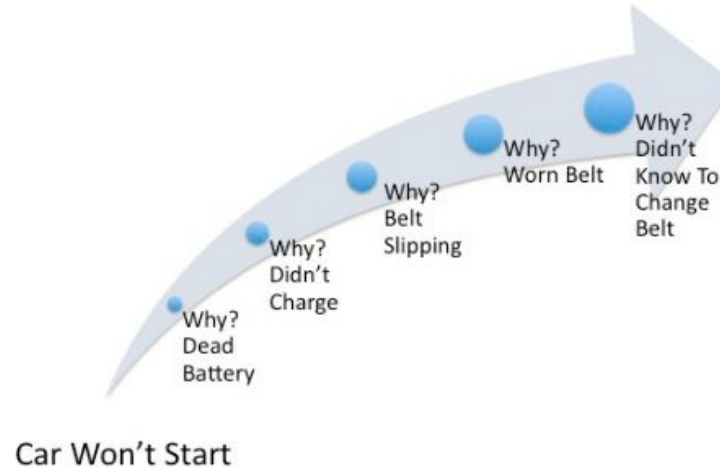
But



oms!

Fix it! - How find the ROOT cause

5 Why Analysis



Fix it! - Keep a Record of What You've Tried

Daybook

- ★ it's helpful to maintain a daybook in order to **record notes** from meetings, **design sketches**, a **record of the steps necessary** to install a piece of software
- ★ If you prefer to keep your notes electronically or using a personal wiki.



Fix it! - PUT IN ACTION

- ★ Bug fixing involves three goals:
 - Fix the problem.
 - Avoid introducing regressions.
 - Maintain or improve overall quality (readability, architecture, test coverage, and so on) of the code.
- ★ **Start from a clean source tree.**
- ★ Ensure that the tests pass before making any changes.
- ★ Work out how you're going to test your fix before making changes.
- ★ **Fix the cause, not the symptoms.**
- ★ Refactor, but **never** at the same time as **modifying functionality**.
- ★ **One logical change, one check-in.**

Reflect it!

Learn the lessons of the bug.

- Where did things go wrong?
- Are there any other examples of the same problem that will also need fixing?
- What can you do to ensure that the same problem doesn't happen again?



Reflect it! - PUT IN ACTION

- ★ Take the time to perform a root cause analysis:
 - At what point in your process did the error arise?
 - What went wrong?
- ★ Ensure that the same problem can't happen again:
 - Automatically check for problems.
 - Refactor code to remove the opportunity for incorrect usage.
 - Talk to your colleagues, and modify your process if appropriate.
- ★ Close the loop with other stakeholders.

Chrome Canary (The experimental Chrome)



Console Debugging

The screenshot shows a web browser's developer console with the following content:

Snippets

```
56 // Timing
57 console.time('checking time');
58 fetch('https://api.github.com/users/google').then(data=>data.json()).then(data=>{
59   console.timeEnd('checking time');
60   console.log(data);
61 });
62
63
```

Console

top Filter Default levels Group similar

Hello world

Everything is fine.

Ohh nooo!

shit!

Gonzalo: 1

(index)	name	age
0	"Pulguita"	4
1	"Nala"	1
2	"Rocky"	2
3	"Plutarco"	1
4	"Angelito"	3
5	"Lobo"	4

Array(6)

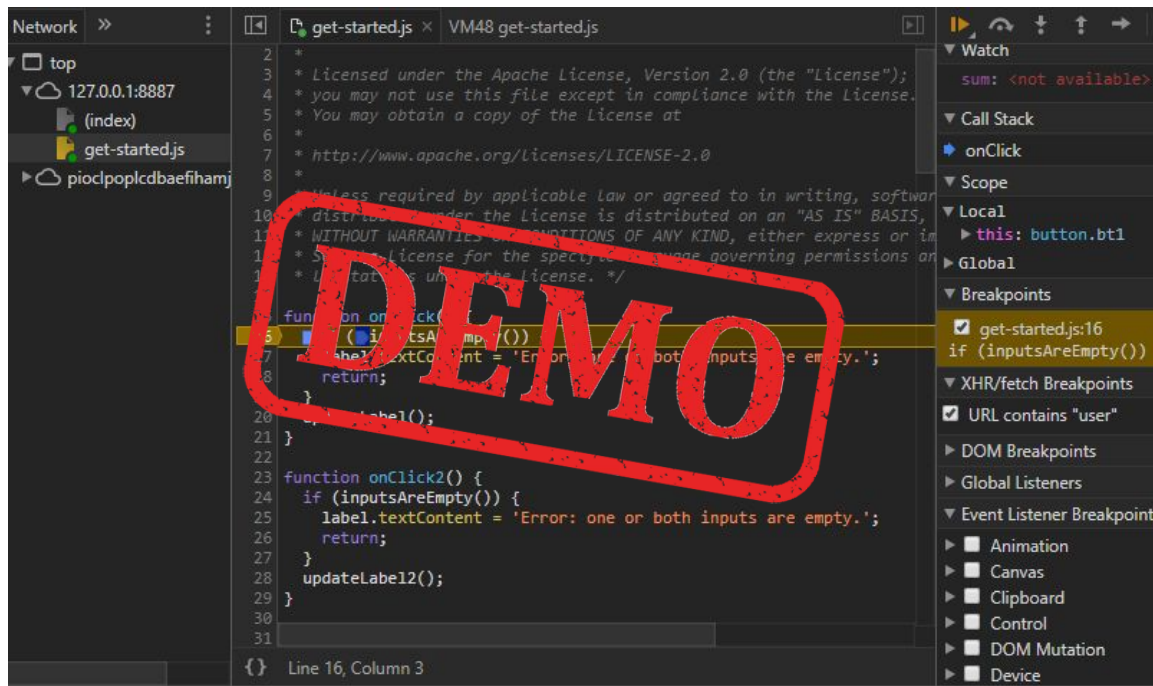
undefined

checking time: 615ms

DEMO

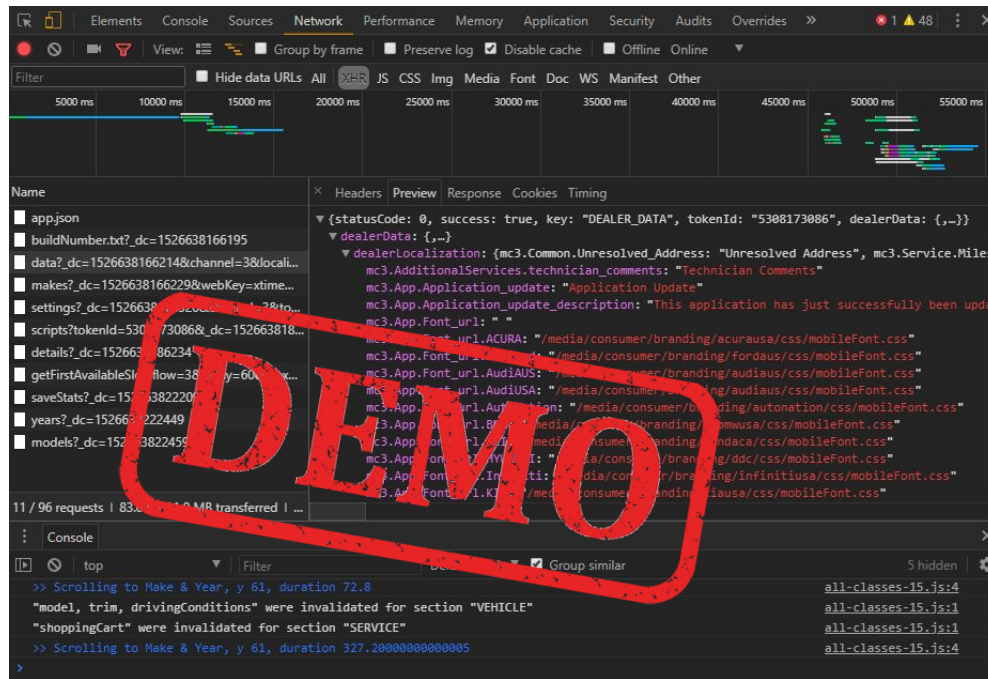
Source Debugging

- Line-of-code breakpoint
- Conditional line-of-code breakpoint
- Dom
- XHR
- Event Listener
- Function
- Snippets
- **Filesystem**

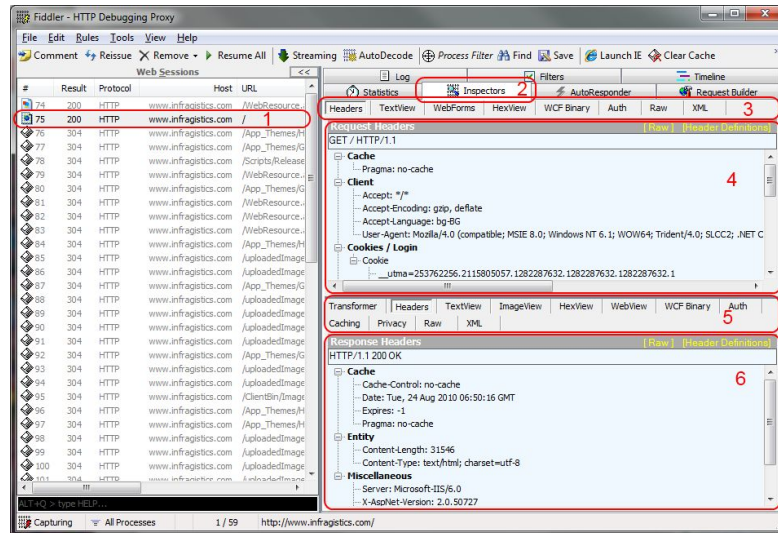


Proxy/Network Debugging

- Proxy debugging
- Network availability
- Timing
- Filtering
- HAR files (HTTP archive)
- HAR viewers
- HAR analyzer

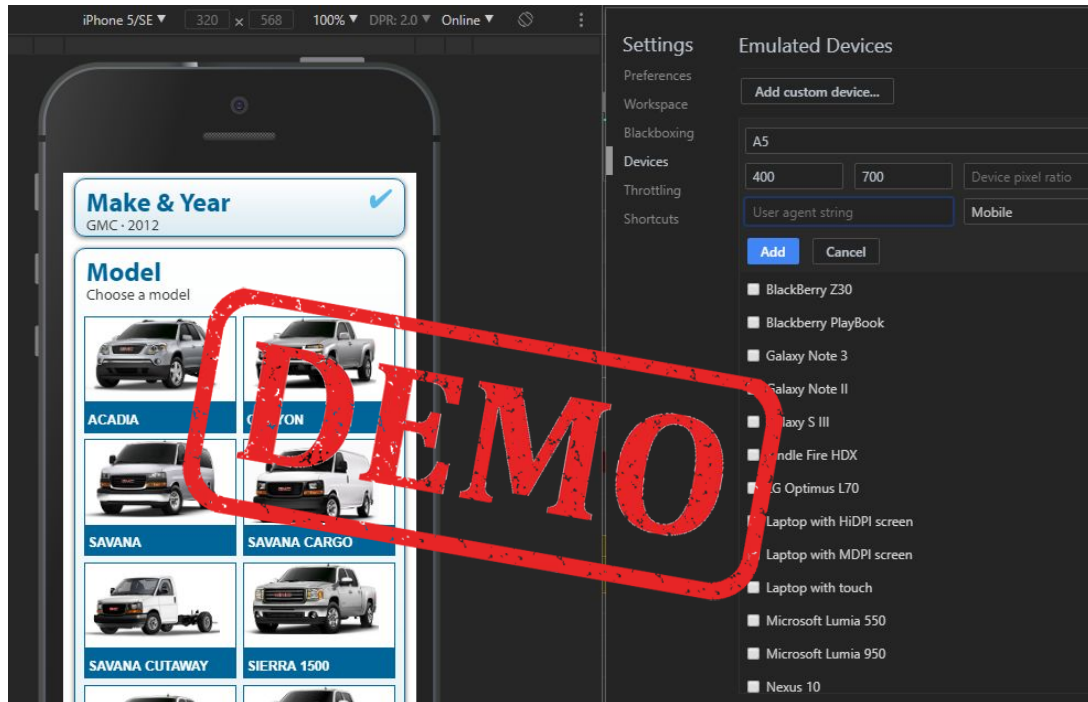


Proxy/Network Debugging with Fiddler and Charles

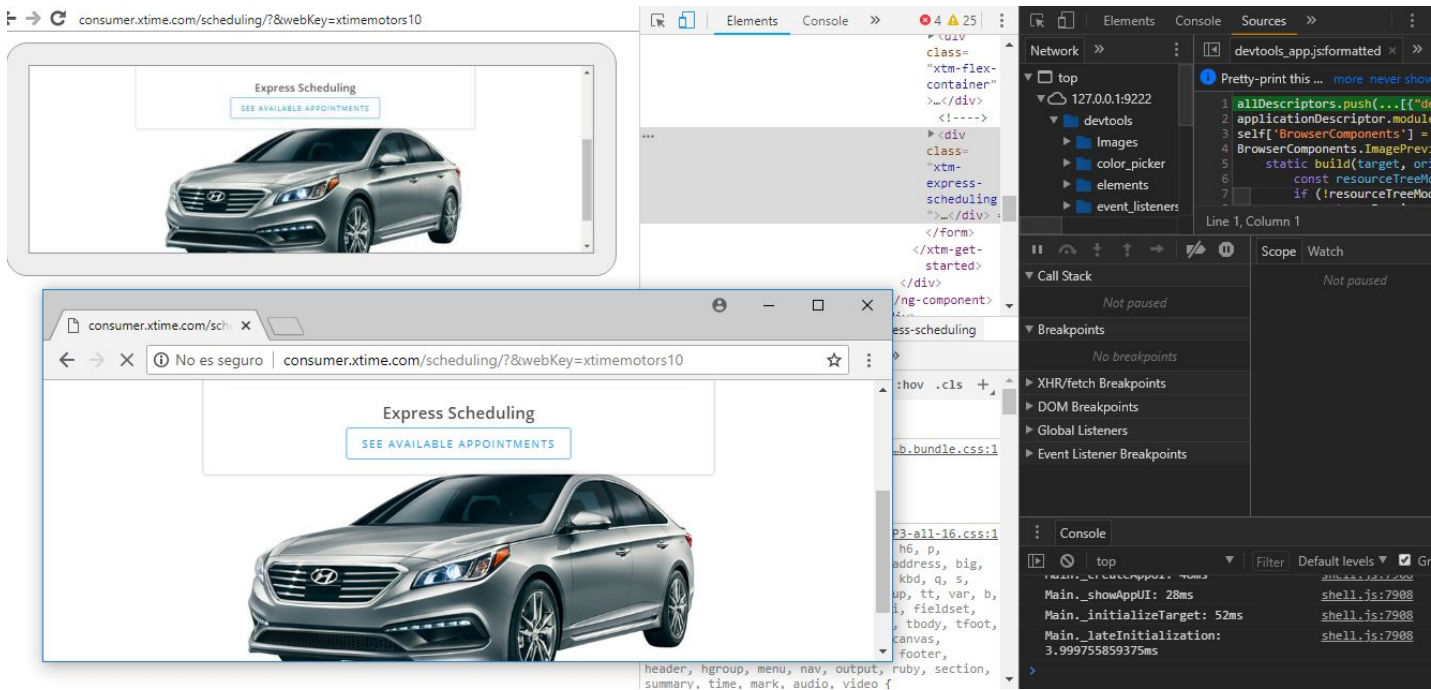


Device Emulation

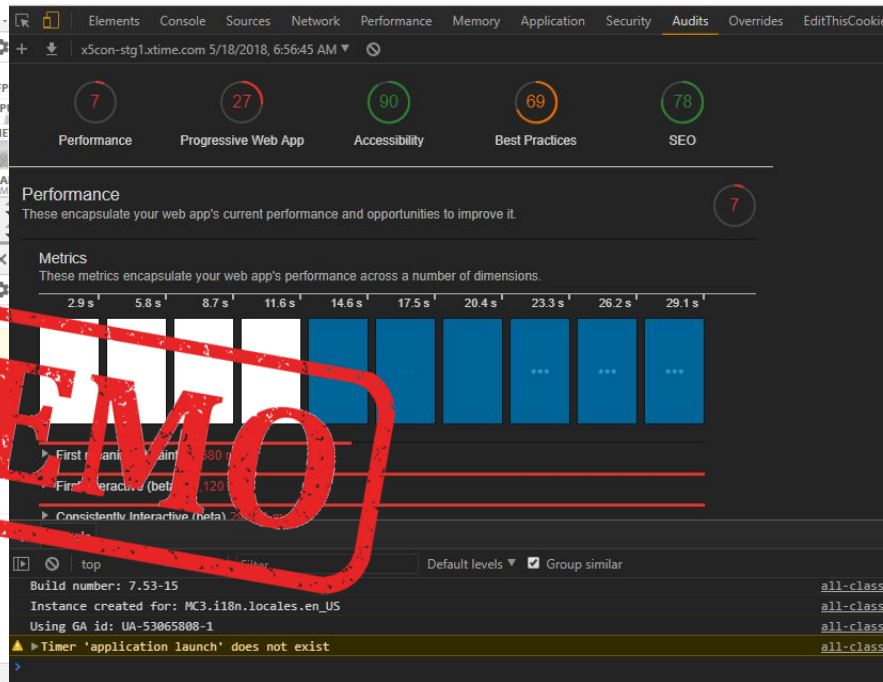
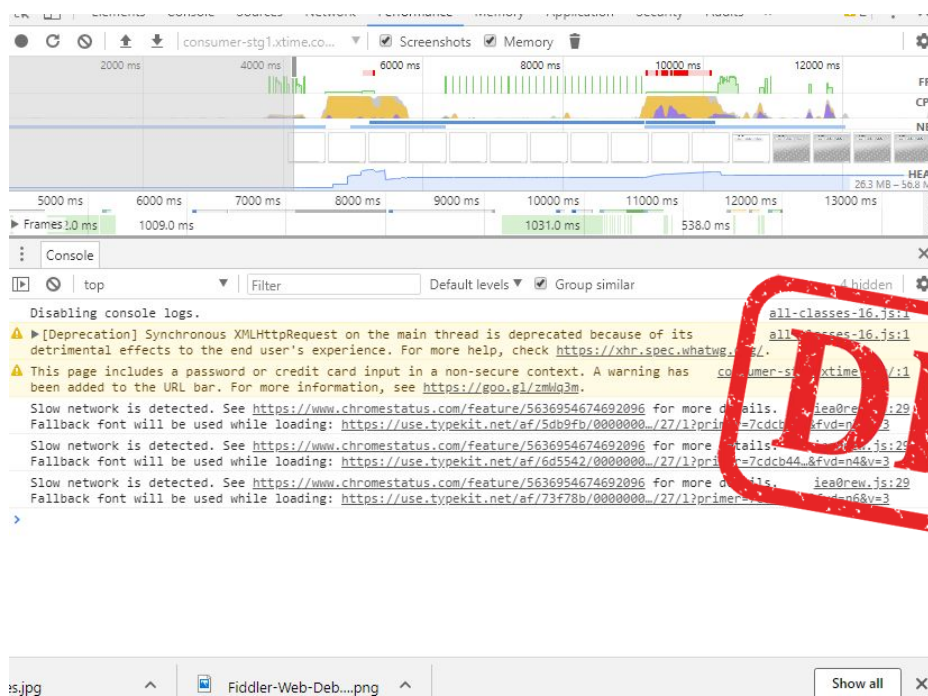
- Simulate any device
- Test in multiple resolutions



Remote Debugging



Performance and Audits

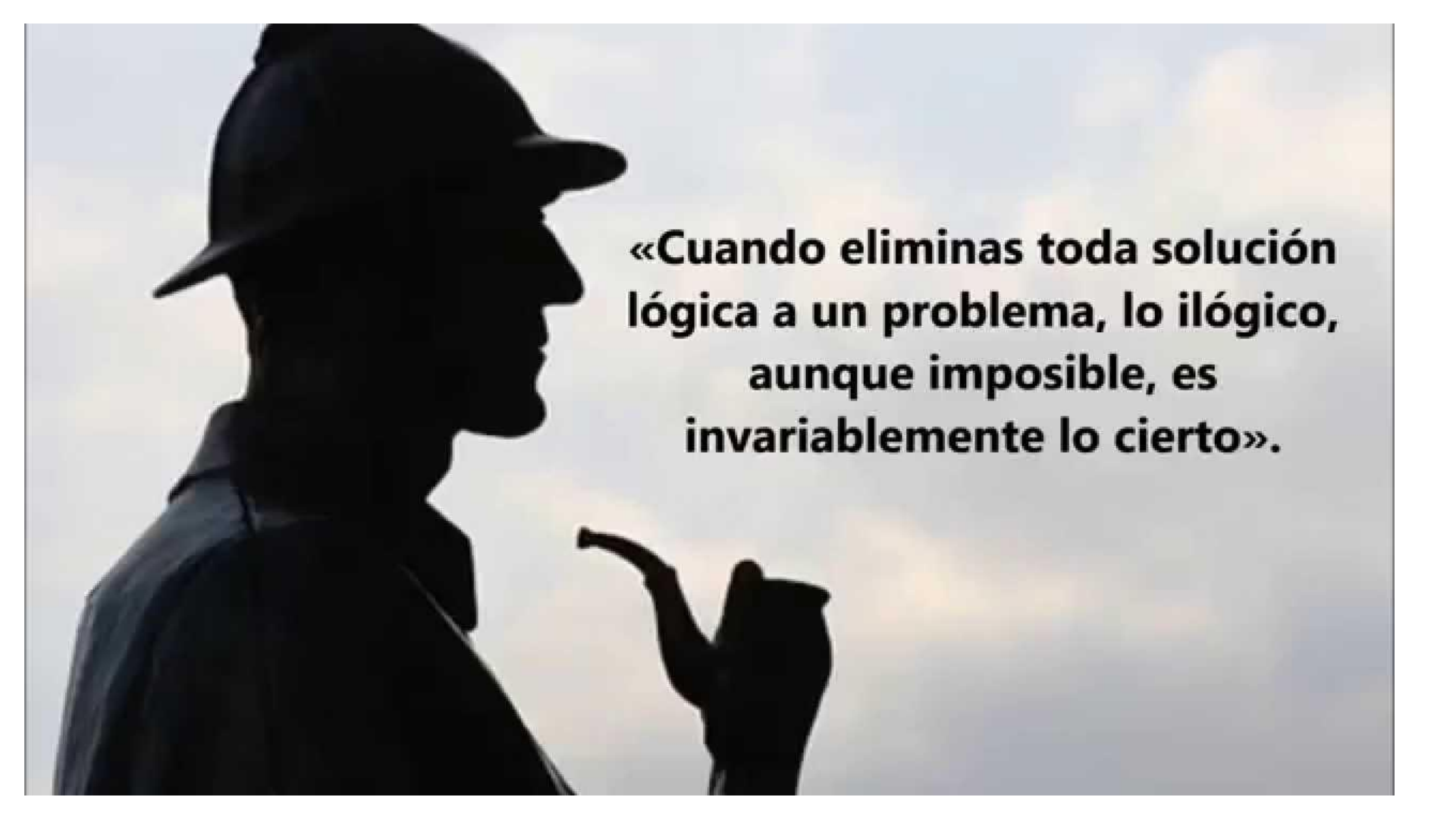


More info!

- “Debug it” book

https://www.e-reading.club/bookreader.php/142038/Debug_It!_Find,_Repair,_and_Prevent_Bugs_in_Your_Code.pdf

- Chrome devtools official documentation <https://developers.google.com/web/tools/chrome-devtools/>
- Chrome devtips <https://umaar.com/dev-tips/>
- Awesome-chrome -devtools
<https://github.com/ChromeDevTools/awesome-chrome-devtools>



**«Cuando eliminas toda solución
lógica a un problema, lo ilógico,
aunque imposible, es
invariablemente lo cierto».**

Gracias!

