Debugging Techniques for Drupal

TexasCamp 2016

https://www.texascamp.org/sessions/debugging-techniques-drupal-and-other-web-applications

Rob Ristroph
Technical Architect, Acquia
@robgr

History

- My first Drupal Camp talk (in Dallas!)
- Given numerous times after that
- Dustin Younse (https://2015.badcamp.net/session/debug-drupal-devel-xdebug-more), Allan Chappell (General Redneck) http://generalredneck.com/sites/default/files/static-content/drupal-corn-2013-debugging-drupal/#/ have given versions of this talk
- You can too -- spice it up with some personal stories and give it to meetup group

Outline

1. What is a Bug
2. What is Debugging
3. Why it is Important
4. “Scientific Method” Approach
5. Toolbox
6. Other tricks
7. More reading

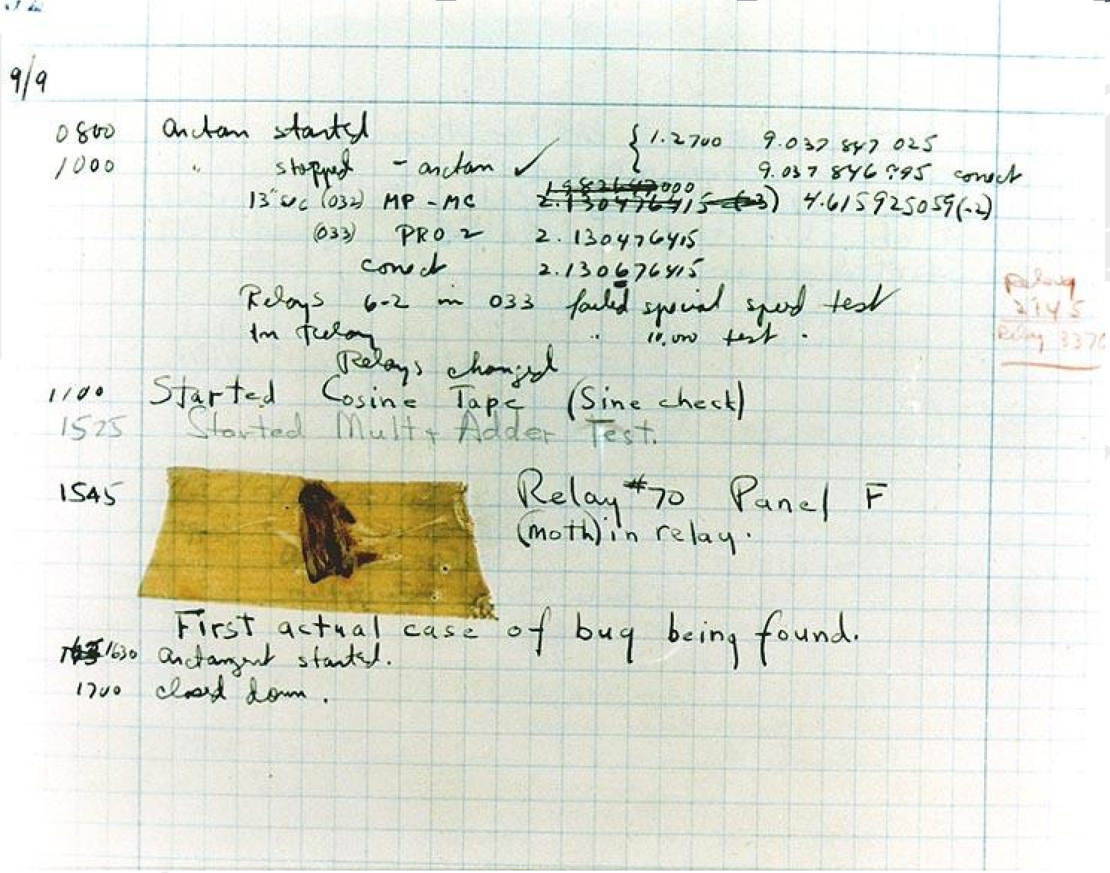
What is a Bug

Your mental model of the code and it’s actual behaviour don’t match.

Usually you typed code that you thought did one thing and in fact it did another - most of the bugs you work on are your own.

Difference from “troubleshooting” . . . .

What is a Bug - A Divergence on Origin



9/9

0800 arctan started
1000 " stopped - arctan ✓ {1.2700 9.037 847 025
9.037 846 995 correct
13"uc (032) MP - MC ~~1.982147000~~ ~~2.130476415~~ ~~(-3)~~ 4.615925059(-2)
(033) PRO 2 2.130476415
correct 2.130676415
Relays 6-2 in 033 failed special speed test
In relay " 11.000 test .
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Mult + Adder Test.
1545 Relay #70 Panel F
(moth) in relay.
First actual case of bug being found.
~~143~~ 1630 arctangent started.
1700 closed down.

Relay 2145
Relay 3370

https://en.wikipedia.org/wiki/Software_bug#Etymology

Debugging is the Process of Making Your Mental Model Match Reality

- Understanding WHY the bug happened is different from fixing it

Why is Debugging Important ?

You spend more time **debugging** than you do **programming**. Furthermore the time debugging is much harder to estimate.

Why is Debugging Important ?

“As soon as we started programming, we found to our surprise that it wasn't as easy to get programs right as we had thought. Debugging had to be discovered. I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs.”

--Maurice Wilkes, 1949, developing the first stored program computer

Why is Debugging Important ?

- You do it more than you realize.
- It’s the source of much uncertainty in estimating and delivery.
- As a distinct thought process / skill, it is possible to become good and more efficient at it.

“Scientific Method” Approach

1. Observe (collect data, as much as possible)
2. Make a testable Hypothesis (change to your mental model)
3. Collect data from the test
4. Adjust understanding (model), goto 1

What Exactly is Broken ?

- Is something not showing up ?
  - New content - is it published ? Front end cache ?
  - Old content - permissions set properly, or changed ?
- Is something showing up that shouldn't ?
  - Raw html or javascript in a wysiwyg field ?
- A more complex behavior - workbench or etc - can we state exactly the steps to cause the bug, and why it's not what we expect ?

Note - non technical members of your team have huge impact collecting data at this stage.

Replicate the Bug

- User reports matter
- Worst case is making changes, waiting to see if the customer reports the problem is still there
- Replication can be tedious, but extremely valuable
- Observe and think about your user's operating procedure
- Without being able to replicate the bug, **you can't debug.**

Sometimes figuring out how to replicate the bug is 99% of fixing it.

Work From the Bottom Up

- Log files
  - Know where they are on your systems / environments
- multitail
  - Linux / Mac utility to easily view logs, with more options
- Contextual information - browsers, environments, users

Vacuum up as much information as possible in the first stage.

Where is it Broken ?

- Custom Module
- Theme template.php
- Theme template
- Configuration in database

Potential tests - disable modules, switch themes, re-install clean without live data.

Divide-and-conquer by narrowing down where the mental model breaks.

Debugging as Scientific Method Iteration

- Change **ONE** thing at a time
- Test that change
- Repeat - *Undoing the change if it gave no information*

Better debuggers are generally better at thinking of clever changes and tests.

- “Cheap” tests first (clear caches, etc)
- Test for common problems first
- A good test should narrow the problem scope by eliminating something

Git is your friend

- Save your progress as you work
  - Re-create your Features
  - Quickly un-do unhelpful changes
  - Makes Rabbit Holes manageable

Better debuggers generally take notes and keep a log.

Always undo experiments as you work (reload DB, etc).

Git diff is your friend

- Remove debug statements
- Ensure you only changed as much as needed

“Database diff” -- hard but dumping “drush vget” can help.

Configuration in files (Features, D8 yml files) help enormously.

Better debuggers generally take notes and keep a log.

Git blame is your friend

- Who wrote (committed) offending code
- Should **NOT** be a witch hunt
- **Should** be a chance to understand the context of the code
  - Re-reading the old Jira tickets or other requirements can cause you to re-assess everything

Find the context of the code where the problem is.

Use "git annotate" in politically sensitive situations.

Make the Future Easier

- Watchdog (D7)
- \Drupal::Logger() (D8)
- syslog module
- http://loggly.com
- Write a test !

Thoughtful instrumentation of your code as it's written the first time can massively pay off later.

Inspection Tools

- Backend - watchdog(), print(), dsm()
- XDebug - have configured in advance
- Frontend - learn the browser dev tools
  - Javascript step-by-step debugging
  - Asyncronicity issues are more common

Sophisticated tools should be set up beforehand, and used in development as well as debugging.

Generally, structured approach and thinking about tests / hypotheses pays off better than fancy tools.

“Interaction” Bugs are the Hardest

The hardest bugs are those that only appear when two “bug free” components interact.

- Module weights, order of hook operations
    - Systematically disable modules, change weights
- Theme / module interactions
- External service requests

If your problem resists divide-and-conquer, maybe it’s not in one component or the other, but in how they connect.

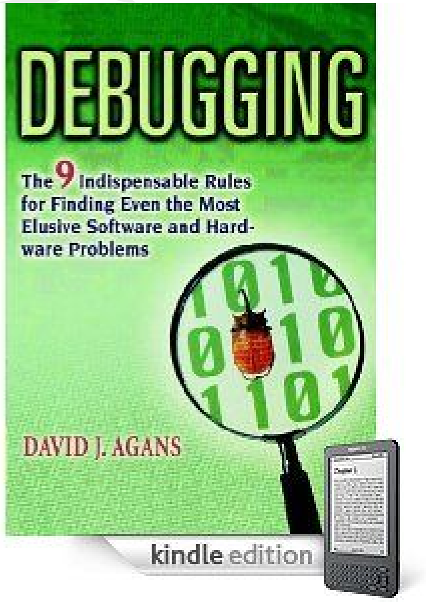
Performance Related Debugging

- Just like other debugging:
- Replicate the problem ! Otherwise you flail at random
  - Apache bench (ab), wget spiders, load generators
- Add headers, log statements, to indicate cache hits / misses
- Different logs often apply - mysql or system logs

Further Reading (and free book!)

“Debugging: The Nine Indespensible Rules” by David J. Agans
http://www.debuggingrules.com/

1. Understand the System
2. Make it Fail
3. Quit Thinking and Look
4. Divide and Conquer
5. Change One Thing at a Time
6. Keep an Audit Trail
7. Check the Plug
8. Get a Fresh View
9. If You Didn't Fix It, It Ain't Fixed


DEBUGGING
The 9 Indispensable Rules for Finding Even the Most Elusive Software and Hardware Problems
DAVID J. AGANS
kindle edition

Conclusions . . . .

- Thinking strategically is more important than applying fancy tools
- The hardest bugs are “Interaction” bugs

Finally . . .

Debugging can be hard to tell someone how to do, but it can be learned if you persist and think about it. Level up !

Debugging Techniques for Drupal

TexasCamp 2016

https://www.texascamp.org/sessions/debugging-techniques-drupal-and-other-web-applications

Rob Ristroph
Technical Architect, Acquia
@robgr