Debugging Techniques for Drupal

(and other web systems)

Robert Ristroph

Email & AIM: robert@fourkitchens.com

Google Chat: rgristroph@gmail.com

Twitter: @robgr

Slides: http://drupalcampcharlotte.com/sessions/debugging-techniques-drupal-and-lamp

FOUR KITCHENS

Outline

I. Why Debugging is Important
II. Structure "Scientific Method" Approach
    I. A bug is a model vs. reality mismatch
    II. Hypothesize
    III. Test
    IV. Refine and Repeat
III. Bag of Tricks
    I. Clever selection is key

More of your life is debugging than coding

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

What is Debugging ?

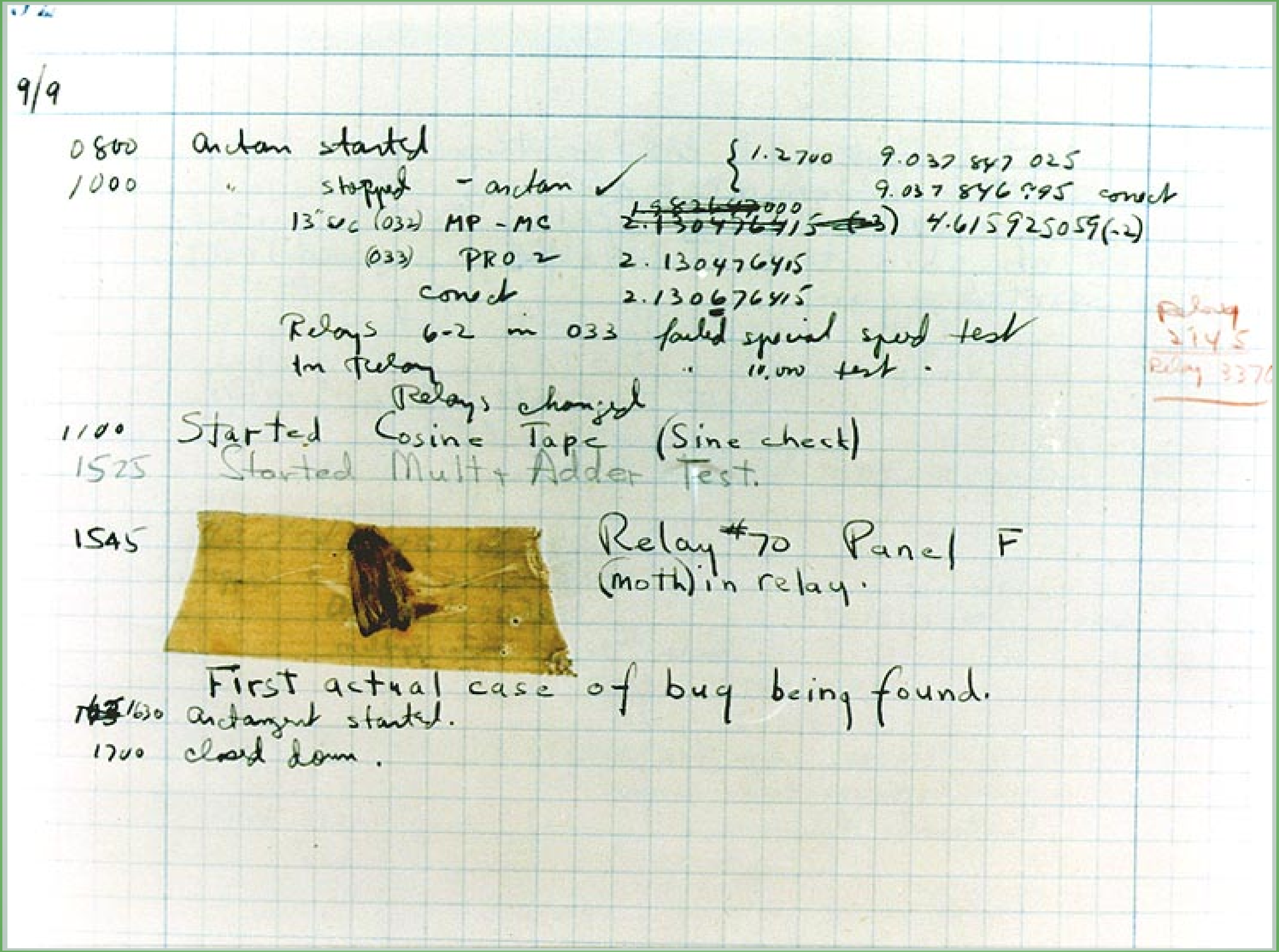
- Not simply good development practices
- Not making random changes (more bugs)
- Not making changes because of a clearer specification
- Even fixing the bug, once found, is not different from other coding

Debugging is developing an understanding of a coding error, so that the behavior of the code and your expectations of it match.

Bugs are Programmer's Errors

Bugs don't fly in and creep into code in the night, or appear as the 0s and 1s get old and rusty. A programmer puts them there – keep this in mind if you want to produce fewer bugs over time.

By the time you are debugging, a bug is a mismatch between your mental model of what the code should do and what it does.



9/9

0800 Anctan started

1000 " stopped - anctan ✓ {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 " 10,000 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 Antangent started.

1700 closed down.

Relay 2145

Relay 3370

Hypothesize and Test

(1)Gather initial info (logs, screenshots)
(2)Make a hypothesis
(3)Test it, analyze
(4)Refine Hypothesis, go to (2)

- Similar to the scientific method
- Can be learned but not taught

·The key is making hypotheses that are easy to test, and also help your understanding of the code.

Gather Information

- Learn the location of all logs – look at lesser known (mysql)
- Watchdog / syslog
- Teach your users how make good bug reports
- Make sure your user knows how to take a screen shot

Project and account managers, or other non-developers, can be *extremely valuable* to your organization by getting good bug reports

Replicate the Bug

- User reports are important
- Worst case is making changes, waiting to see if the customer reports the problem is still there
- Replication can be tedious, and take as much insight as any part of debugging
- Observe and think about your user's operating procedure
- Without being able to replicate the bug, you can't debug

Cheap Tests First

- Clear all caches (browser, Drupal, Varnish, any other) (not a fix, just a test)
- Check that the right version of the code is on the server
- Change to a default theme
- Check for old js / css being delivered from a CDN
- Remember all the dumb mistakes you have made, and check for them . . . “check the plug”

Someone Else Solved It ?

If simple checks haven't found it, check d.o issue queues and the web.

- Art to searching – use exact error messages, except for node numbers and etc
- Ask on #drupal-support
- Try dev version of modules (be sure you can undo)
- Always post – to describe your problem on related issues, to confirm when you have solved it

Common Problems

- Permissions on an input filter
- Rebuild node_access table
- Permissions on imagecache and aggregated js/css files (whole /sites/all/files)
- Content Permissions is giving access to some CCK fields and not others
- Cron related (cron_semaphore)
- Keep your own list (perhaps per-project)

*List-checking for cheap and common bugs can't replace **thinking**.*

Uncommon Problems

- Rare and obscure problems are useful mainly as “war story” material, and teaching debugging.
- Lists of cheap tests and common bugs should be short
- Quickly get into “thinking mode” and hypothesize – test – analyze – repeat

Inspection

- Krumo / devel module
- print(“<pre>”.$stuff.”</pre>”);
- print(“<pre>”.print_r($stuff,TRUE).”</pre>”);
- Use watchdog() for values on pages that refresh or are in a cron hook
- xdebug / breakpoints / code tracing – great but don't let it stop you from thinking of good tests (new golf clubs vs. golf lessons)
- Have a standard debugging setup as part of your development (before debugging) (try Quickstart project)

Inspecting the Database

Always keep a copy of the DB in one state, do experiments, the compare (then restore to original state)

- Having a “starting point” db copy can be key to replicating problems
- Use drush to dump variables to files, then diff
- Try not to end up blindly scanning the differences; develop hypothesis / guess / theory and look for that specifically

Devel Module

- Node access / permissions block for debugging
- User switcher for recreating problems (have a test user for every role)
- Look at query list
- Devel features are a kind of “common problems” list

/devel/php as a way to do quick tests – check if a PHP extension is installed; or unserialize a cut-and-paste from the db; test code snippets for expected behaviour.

Theme Debugging

- No default template file when providing a lower level one(?)
- Devel module themer module
- Limits on the number of CSS files - IE especially - things only work when you use css aggregation
- hook_form_alter() vs hook_form_FORMID_alter, and module weights
- Typography, and even color profiles in images are handled differently in different browsers, platforms

Performance

•Learn how to use “ab” (apache bench)
•Wget spiders
•Load Storm (commercial service)
•MySQL logs and innotop
•Cool trick: put a header indicating a cache hit or miss, that you can look for with the LiveHTTPHeaders Firefox installation
•Use of xhprof and similar needs to be targeted at a problem

Like any other debugging, but replication is complicated and inspection tools different.

“Interaction” Bugs are the Hardest

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

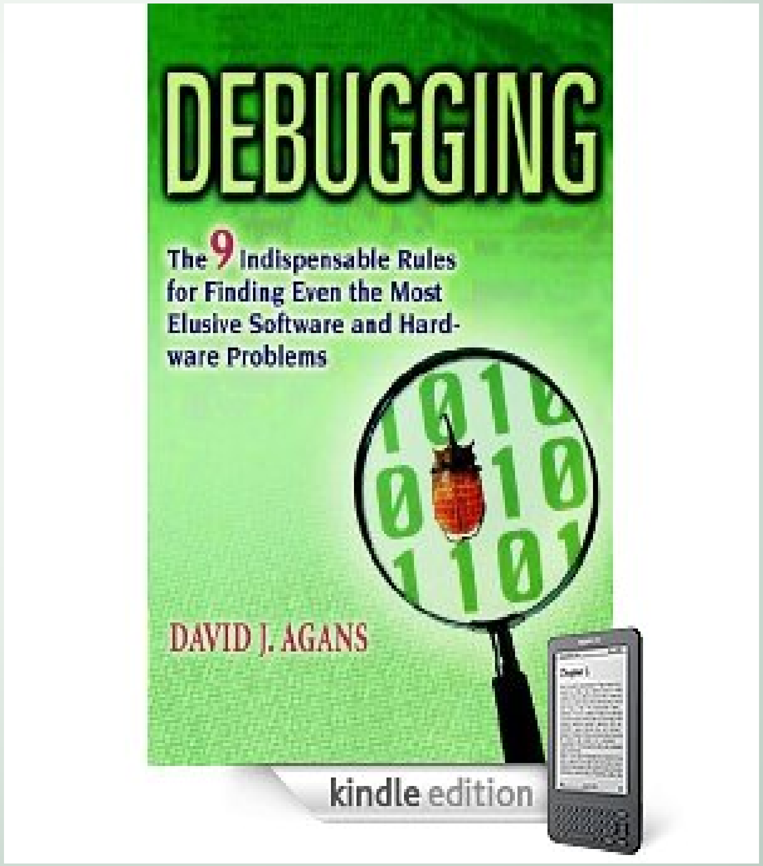
- Module weights, order of hook operations
- Theme / module interactions
- External service requests
- Mis-use of APIs
- Unexpected cache clear causes performance issues

If your problem is resistant to being narrowed to a certain component, think along these lines.

Further Reading

"**Debugging: The Nine Indispensible 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

Conclusion

- Coming up with the right guesses and tests is more important than fancy tools
- The hardest bugs will be ones that are caused by interactions

If you pay attention to how you debug, you will get better at it.

Expert debugging is not hard to learn if you try, but is hard to teach; there is no step-by-step recipe.

DrupalCamp Charlotte Survey:
http://bit.ly/KjPvhU

we make BIG websites

FOUR KITCHENS

cc