Pessimism, Entitlement, and other virtues of a successful debugging approach

by Allen Shaw, for JOINERY
http://JoineryHQ.com
@JoineryHQ

Slide 2

Email:

> OMG, that didn't work. I tried the new feature and I can't do anything.
> **Can you fix it please, thanks!!**

You may sometimes get a request like this.

This is a good thing. It could be better. It could have more information. It could be a lot more specific. But it alerts you there's a problem. And solving the problem begins with awareness. In this talk we'll discuss ways to get from this starting point to a resolution and a happy client.

Slide 4: Related talks

These are two great sessions on debugging I heard recently. I've borrowed from them here and there because they're great.

Robert Ristroph's is an excellent discussion of the logical principles of debugging. I think some of you were in that talk. Lucky you.

Gary Hockin's talk covered some great examples of using PHP's Xdebug extension in conjunction with PHP Storm to conduct "step-through debugging," which I'll demonstrate a little later. This was the LoneStar PHP last weekend in Addison, where around 250 PHP professionals took in two days of talks on a wide range of topics. If you want to keep improving as a developer, you should go to LoneStar PHP next year.

Slide 4: Debugging is hard

Here’s a quote that Rob Ristroph shared in his talk.

Slide 8: Act 1

Act 1. Disappearing nodes!

Allen's client David, membership director for a charity service organization, emails to say that nodes are disappearing from his site. Two that were there yesterday are now gone, as is one that was definitely there last week. Alarmed, Allen checks the site’s recent log entries and finds no node deletions recorded in the past 24 hours. He then compares the current node list to last week's backups, and finds lots of nodes that have been deleted -- but is unsure if any of them are relevant.

Allen asks David if he can remember which nodes were deleted; David names three nodes; each one was an entry for a local chapter of David’s organization. Allen checks the list of nodes deleted in the last week -- and none of David's missing nodes are listed there. Allen then checks /admin/content and finds the nodes still exist. Why does David think they're deleted?

Since David's arrival at the organization 9 months ago, he's been using a views-generated page as a calling list to check in on local chapters. It's called "Newest Chapters" even though the organization doesn't often add new chapters. Recently some chapters have been dropping off the list.

Allen explains to David that this view shows only chapters that have been added in the last year, and since these three chapters were added just a little over a year ago, they've just now rolled off the list. Because this organization rarely adds new chapters, David had “gotten away” with using it as a comprehensive chapter list. Allen points him to the "All Chapters" page, and David is happy again.

Question: How could Allen have been more efficient in solving this issue?

Slide 9: Act 2

Act 2. Can't save new content!

Allen's client Sarah launched her new e-commerce site last month; she calls Allen to say that something's wrong because she can't save new content. She's tried and tried and it looks like it should save when she clicks "Save," but then it's just not there. She's quite pressed for time and stressed out, so Allen says he'll look into it and email her when it's fixed.

Allen then tries to reproduce the issue by creating some pages on the site, which works just fine.

He emails Sarah and asks for more details. Sarah reports that she's continued trying to save the new page, and is growing more frustrated because she's on a deadline to publish an article about the upcoming promotion. No matter what she tries, her page at first appears to be saving, but then is not to be found in the "Hot Deals" section listing.

Allen calls Sarah and walks through the node creation on the phone. At this point they realize that Sarah has been forgetting to assign the "Hot Deals" taxonomy term to the created node, so although the page is created, it's not listed where Sarah expects to see it. Sarah smacks her forehead and publishes the "Spring specials" article, and Allen heads over to /admin/content and deletes the 47 copies of it that Sarah had created.

Slide 10: Act 3

Act 3. I can't masquerade

Allen's client Jim uses the Masquerade module when conducting user trainings and support at his company. One day Jim emails Allen to say he can't masquerade as anyone; he needs that functionality, but wants to be sure this fix doesn’t derail other development tasks that Allen is already working on for him. Allen spends a couple of hours on the live server slogging through the Masquerade code using vim, to look for a solution, but can’t make much headway and eventually turns to the other development tasks, because these still need some work and are due next week.

By the time those are completed, Jim has had to reschedule several training sessions, so he inquires again about the Masquerade fix.

Allen finally turns to Google, and then to the Masquerade issue queue, where he finds a regression bug introduced by a recent security update. Allen recalls that the security update was applied to the live system a couple of weeks ago, so this is likely the issue. Allen notices that the fix for this issue has already been committed and released in the next Masquerade version, which was released just two days after the security release that contains the bug.

Allen runs `drush up masquerade`, and the problem is fixed.

Question: What was Allen’s biggest mistake?

Slide 11: Act 4

Act 4. Newsletter emails aren't going out!

Allen's client Karen works at a small non-profit organization and contacts Allen for help because her CiviCRM newsletter emails aren’t being received by her constituents.

Thinking of all the things that could prevent email delivery, Allen gets to work. He checks the mail server reputation to see if it's blacklisted; he fiddles with the postfix configuration; he restarts postfix; he reboots the server; something has to work.

Each time he changes something, he tries sending a test mailing, which requires clicking through several configuration screens. It's time-consuming, but he's a hard worker.

Finally he checks the Scheduled Tasks page and finds that scheduled tasks have not been running for two weeks. Something must be wrong with the cron job that triggers Scheduled Task processing, so he tries a couple of changes in the crontab file, but still no luck.

This crontab entry is written to use a use of a specific Drupal user to trigger the Scheduled Tasks, and after examing crontab, Allen finds that the username being used for this job belongs to a user who no longer exists on the site -- ah, success!! Allen recreates and re-configures the user account so that the cron job will run properly.

He then emails Karen to tell her that the problem has been found and fixed. Karen is relieved and writes back a note of thanks.

Three days later, Karen writes back to say that the emails still haven't gone out. What's going on, she asks?

Allen checks the crontab entry and finds that in all his changes to the cron command, he had introduced a typo, so that the cron job had actually not been running at all.

Allen corrects the typo, says a little prayer, and emails Karen that all is well.

Question: What the hell is this guy thinking?

Slide 12: That sucked.

This guy is a hard worker, and very sincere, and very interested in solving problems. But this kind of ad-hoc seat-of-the-pants scrambling is not going to be the best way to find and fix bugs or perceived bugs. So what are some appropriate responses?

Slide 14: Skepticism

Some part of me should assume that there is not really a bug -- until I can see it happening myself. Without seeing it happen, I have no idea if:

- The problem exists

- For whom it exists and under what circumstances
- What the nature of it really is

I can guess and assume, but that wastes a lot of time.

We need to replicate the Bug. Otherwise, don't have a bug, we have a rumor. In Rob Ristroph's talk, he says, "Without being able to replicate the bug, you can't debug." If you can't replicate it, how will you ever know that you've fixed it?

When David said nodes were being deleted, Allen could have asked a few questions before diving in to hunt around on the server for evidence.

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

Slide 15: Finger-pointing

This response says, "It's probably the user's fault." Or, more kindly, "Is there a mismatch between the program design and the user's expectation?"

Especially on systems that have been working well for some time and on which the codebase has not recently changed, the sudden appearance of a "bug" could mean the user is doing something different. Here's another reason to ask questions and clarify the exact steps to reproduce the problem.

When Sarah reported that nodes couldn't be saved, is it likely that Drupal continues to work properly in other ways but suddenly fails at this, or is it more likely that there's something unusual about Sarah's workflow?

Slide 16: Entitlement

This response says, "Surely somebody else has already fixed this for me."

Once I've determined what the problem is, rather than trying to fix everything myself, I should remember that I'm working on open-source software, and there's a good chance someone else has found and fixed this already.

Google is your friend here.

When Jim found issues with Masquerade, instead of slogging through the Masquerade source code (in an unfamiliar tool like *vim*, on the live server, no less!), a quicker solution path probably involves checking whether similar problems have been seen on any of the other 54,000 sites that run this module.

Slide 17: Impatience

Impatience says, “I don’t have time to keep reproducing this bug.” I’m too busy. My time is too valuable.

When Allen was trying to reproduce Karen’s email troubles, he spent a lot of time clicking through screens to reproduce that bug, and he did it over and over. This is not a good use of time.

So now we know there’s a problem, and there’s probably not an easy fix for it. We’re going to be digging through a number of possibilities, and we’ll need to check repeatedly (that is, more than once) that this is fixed. If there are more than a few clicks involved to reproduce this bug, automate those steps.

Use a tool like Selenium to create a repeatable, scriptable test that allows you to observe this bug over and over, consistently and quickly. As a bonus, when you’re done, you’ll have a test you can use later to ensure this bug never comes back.

We’ll look at that in a few minutes.

Slide 18: Laziness

Laziness says, “Don’t repeat yourself. Do no more than necessary.”

Here’s where a consistent and scientific approach comes in. There are any number of possible causes for unexpected behavior in a program. So we have to narrow the possibilities carefully and methodically.

1. Observe (collect data, as much as possible)
2. Make a testable Hypothesis (change to your mental model)
3. Change one thing to test your hypothesis
4. Collect data from the test
5. Adjust understanding (model), reformulate the hypothesis, and repeat.

Rob Ristroph:

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

When Allen was trying to fix Karen’s email problem, he took a pretty haphazard approach, trying this, trying that, changing multiple things at once without a plan. A more scientific approach would allow him to narrow the possibilities quickly, slicing off entire branches of possibilities as we go.

Slide 19: Pessimism

Here's another that goes hand-in-hand with Laziness and the scientific approach.

This fellow's dealing with a power outage in his office, so he heads to the basement and tries a quick fix. It's gotta be one of these wires here.

What he doesn't realize is that his entire state is experiencing a blackout. None of those wires in his basement is going to fix his problem.

Pessimism says, "Whatever you think is the problem, it's probably bigger than you think." Take a step back. Think of what else -- or who else -- this bug might affect.

This would have helped in debugging Jim's problems with masquerade, because it wasn't only Jim, and it wasn't only on Jim's site -- it was everyone in the whole world who used that version of Masquerade.

Slide 20: Ruthlessness

There are times when you really do have to dig into the code and find what's going on.

Ruthlessness says, "Be thorough, leave no stone unturned, refuse defeat." In this mode, we're ready to rip the code apart, and we need good tools to do it. Most importantly, a good IDE that can help us navigate the code, examine backtraces, and for the most efficient work, step through the code as it's running.

Slide 21: More skepticism

When we think we have it fixed, here's another great time to be skeptical. Skepticism says, "It's probably not really fixed." We know what the bug was (we think), and we think we fixed it, but what else might this bug have broken? Is there corrupt data that needs cleaning up as a result of this bug? Have we tested it from start to finish to be sure that it works all the way through? Have we broken anything else?

When Allen told Karen her email bug was fixed, it was another three days before she found out it wasn't. If he'd tested it from start to finish, he'd have known already.

So, skepticism: It's probably not really fixed yet.

Slide 23: Logs

Logs are valuable evidence. Don’t let them disappear.

As a general rule: Gather and archive as much evidence as possible as soon as possible after the bug occurs.

Slide 32: Live debugging

Act 5. (Live team work)

Allen receives the following email from his client Bruce:

> Hi Allen,
>
> I need you to look into a problem on the site. I've been having problems uploading a certain image. Every time I try, I get an error that it's not possible.
>
> I added several other images today (see http://debugging.example.local/node/55 for example), but this one is blocked. Is there something wrong with this image?
>
> Thanks,
> Bruce

The site is at debugging.example.local.
Go!