Debugging for beginners

Rob Allen
January 2013
About Me

Rob Allen

http://akrabat.com

@akrabat
Debugging is a methodical process of finding and reducing the number of bugs, or defects, in a computer program thus making it behave as expected.

Wikipedia
Bugs

There are two types of bugs. Trivial and very very difficult.
The 6 stages of debugging

1. That can’t happen.
2. That doesn’t happen on my machine.
3. That shouldn’t happen.
4. Why is that happening?
5. Oh, I see.
6. How did that ever work?

— John Chang, 2003
The debugging process

• Reproduce
• Diagnose
• Fix
• Reflect
Reproduce

Can you make the error happen on demand?
Where to start?

• Don’t trust the bug report!
• Find out what the correct operation is expected to be!
• Only ever work on one problem at a time!
• Check simple things first.
• Ask colleagues about problem area.
Reproduce

- Does it fail on the latest version?
- Does it fail on reported version?
- Match environment as closely as possible.
- Assume user didn’t do as expected.
- Last resort: add some logging and wait for new bug report!
Refine

Reduce the bug to the smallest possible number of steps

If it appears to be non-deterministic, it almost certainly can be made deterministic

Automate the bug - create a unit test!
Diagnose

Investigate the error and work out what has to be done!
Types of errors

The ones PHP tells you about
  Read any error messages and logs
The rest!
  Think & experiment!
Set up PHP to help you!

- Configure php.ini
- Install Xdebug
Useful php.ini settings

error_reporting = E_ALL | E_STRICT
display_errors = On
display_startup_errors = On
html_errors = On
log_errors = 1
error_log = /path/to/php_error.log
Also consider

e: error_prepend_string  = "<div style='clear:both'>"

e: error_append_string = "</div>"
Xdebug

• var_dump() override
• set breakpoints
• inspect variables

Get it from http://xdebug.org (or your distro!)
Xdebug

Ensure Xdebug’s output is always readable regardless of your designer!

Set a custom CSS file in your browser and add this:

```css
table.xdebug-error th,
table.xdebug-error td {
    color: black;
}
```
Types of error messages

- Fatal errors
- Syntax errors
- Recoverable errors
- Warnings
- Notices
- Deprecation notices

Don’t ignore any!
Reading error messages

• Actually *read* the error!
  • it’s usually right
  • backtraces from Xdebug!

• Only worry about the first error
A Fatal error

Fatal error: Call to undefined function datee() in //www/localhost/test.php on line 12
Xdebug display

<table>
<thead>
<tr>
<th>#</th>
<th>Time</th>
<th>Memory</th>
<th>Function</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0011</td>
<td>273416</td>
<td>{main}()</td>
<td>../test.php:0</td>
</tr>
<tr>
<td>2</td>
<td>0.0011</td>
<td>273448</td>
<td>main()</td>
<td>../test.php:3</td>
</tr>
<tr>
<td>3</td>
<td>0.0011</td>
<td>273528</td>
<td>logit()</td>
<td>../test.php:7</td>
</tr>
</tbody>
</table>

Fatal error: Call to undefined function datee() in /www/localhost/test.php on line 12
Exceptions

Zend\Db\Adapter\Exception\InvalidQueryException

File:

Message:
Statement could not be executed

Stack trace:
#0 /www/dev/pristine/vendor/zf-commons/zfc-base/src/ZfcBase/Mapper/AbstractDbMapper.php(141): Zend\Db\Adapter\Driver\Pdo\Statement->execute()
#1 /www/dev/pristine/module/Pack/src/Pack/Mapper/Comment.php(65): ZfcBase\Mapper\AbstractDbMapper->insert(Object(Pack\Entity\Comment))
#2 /www/dev/pristine/module/PensionPack/src/PensionPack/Service/Pack.php(246): Pack\Mapper\Comment->save(Object(Pack\Entity\Comment))
#3 /www/dev/pristine/module/PensionPack/src/PensionPack/Service/Pack.php(251): PensionPack\Service\Pack->addComment(61, 'Pack updated', true)
Exceptions

Look for a previous exception!

$previousException = $e->getPrevious();
The other types of error

- Logical errors
- It doesn’t do what the user expects

These are solved by experimentation and investigation
Var Dump Debugging

Quick and easy:

`var_dump($comment);
exit;`
Var Dump Debugging

object(Pack\Entity\Comment)[932]
protected 'id' => int 0
protected 'pack_type' => null
protected 'pack_id' => int 61
protected 'created_by' => int 1
protected 'user_name' => string 'Rob Allen' (length=9)
protected 'user_email' => string 'rob@akrabat.com' (length=15)
protected 'is_system_comment' => boolean true
protected 'comment' => string 'Pack updated' (length=12)
protected 'datetime_created' => null
protected 'datetime_updated' => null

(If you don’t have xdebug, then wrap in <pre> tags)
Divide and conquer

• Find halfway in process and inspect at that point
• Find halfway in correct half and inspect there
• etc.
Divide and conquer via git

- find a known working commit hash
- `git bisect` until you find the commit that caused the problem
- Read the diff carefully.
Choose logical check points

e.g.

- Test values sent into script
- Test storage
- Test retrieval
- Test display
Step by step with Xdebug

• Add xdebug_break() when you want stop.
• Run in browser
• debugger will kick in when break point reached.
MacGDBp

```
$comment = new Comment();
$comment->setPackType('pension');
$comment->setPackId($packId);
$comment->setCreatedBy($user->getId());
$comment->setUserName($user->getDisplayName());
$comment->setUserEmail($user->getEmail());
$comment->setIsSystemComment($isSystemComment);
$comment->setComment(strip_tags($text));

xdebug_break();
$commentMapper->save($comment);
```
Logging

- Long term error reporting & tracing.
- Different levels for different types of message.
- I use Zend\Log. Also consider monolog.
Zend\Log setup

// setup
use Zend\Log\Logger;
use Zend\Log\Writer\Stream as LoggerStream;
$log = new Logger;
$writer = new LoggerStream($filename);
$log->addWriter($writer);
Zend\Log setup (2)

// Log PHP errors & exceptions too
Logger::registerErrorHandler($log);
Logger::registerExceptionHandler($log);
Zend\Log in use

$logger->log(Logger::INFO, 'My message');

// levels:
//    * EMERG   * WARN
//    * ALERT   * NOTICE
//    * CRIT    * INFO
//    * ERR     * DEBUG
Fix

A quality update is worth the effort!
Know the root cause

Never change the source unless you know why what you’re doing fixes the problem
Clean up first

Start from a clean source tree - save you what you need first

git reset is good for this.
Create your test(s)

1. Add your new test(s)
2. Run them to prove that they fail
3. Fix the bug
4. Run the tests to prove that they pass
5. Run the full suite to ensure no (known) regressions
Refactor

The golden rule of refactoring is to not change functionality.

Therefore refactor before or after fixing the bug.
Commit

If it’s not in source control, then it hasn’t happened.
Reflect

Make sure it doesn’t happen again!
What went wrong?

• Requirements / spec
• Architecture / design
• Construction
• Testing
Change your dev practices?

- Coding standards
- Pair programming / code reviews
- Developer documentation
- Staff training
- Unit testing!
- Refactor
Historical records

- Log every bug in your bug tracker!
Questions?
Thank you

https://joind.in/7813

Rob Allen : http://akrabat.com : @akrabat