Debugging

The why, where && WTF
behind your error messages…

Hunter T.

Phase 1: Day 4

Lecture Topics

- History
- Reading Error messages
- Different Strategies for attack
- Tools
- Cleanup some code (maybe)
- Ten most common errors with examples

Why you care

- 80% of your time as a dev is spent READING CODE

- 60% of that time is spent DEBUGGING

- Good Debugging skills is the one thing that will keep you sane in this jerb…

History

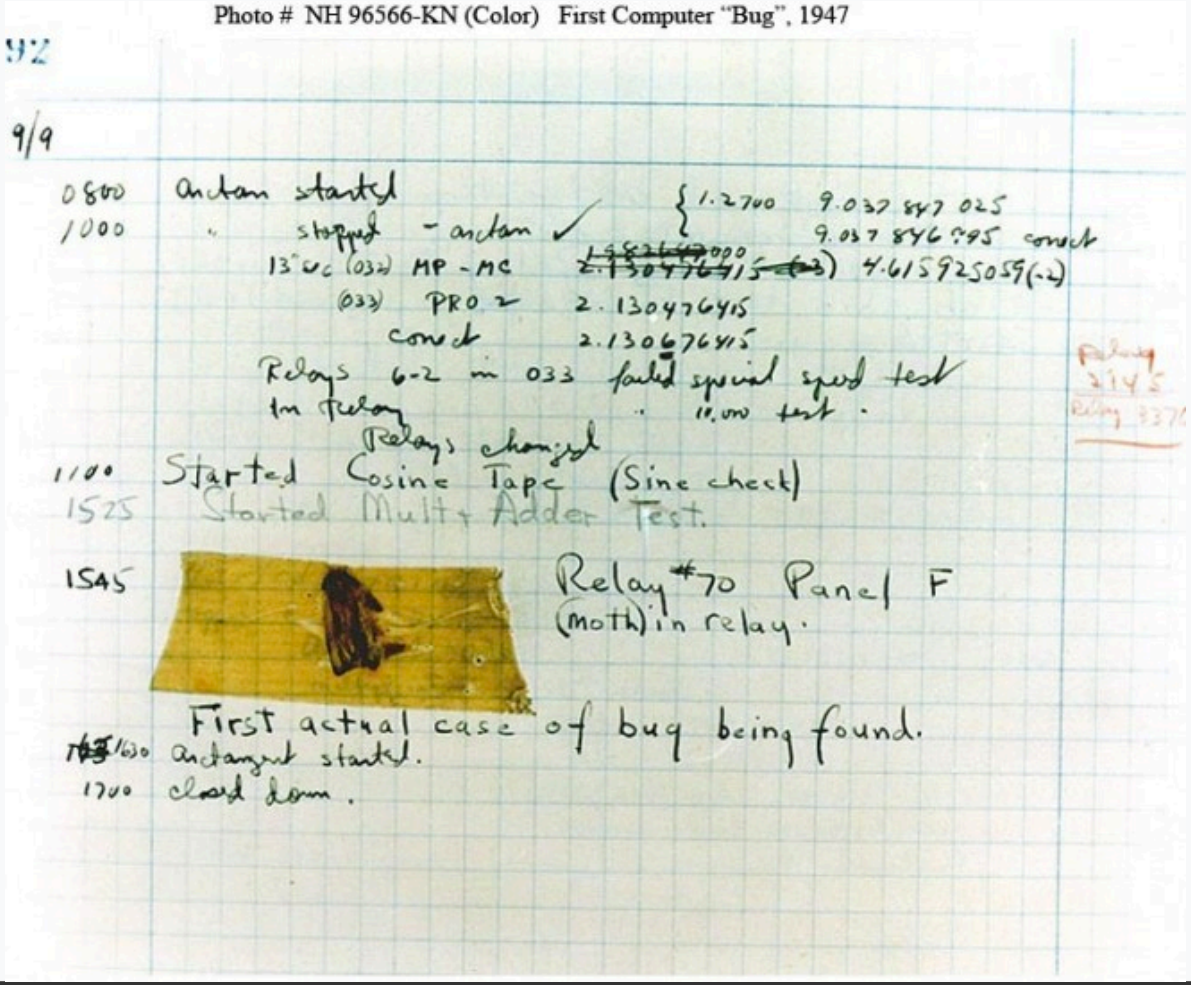


Photo # NH 96566-KN (Color) First Computer "Bug", 1947

9/9

0800 Arctan started

1000 " stopped - arctan ✓

13" uc (032) MP - MC

(033) PRO 2

correct

Relays 6-2 in 033 failed special speed test

in relay " 11.00 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.

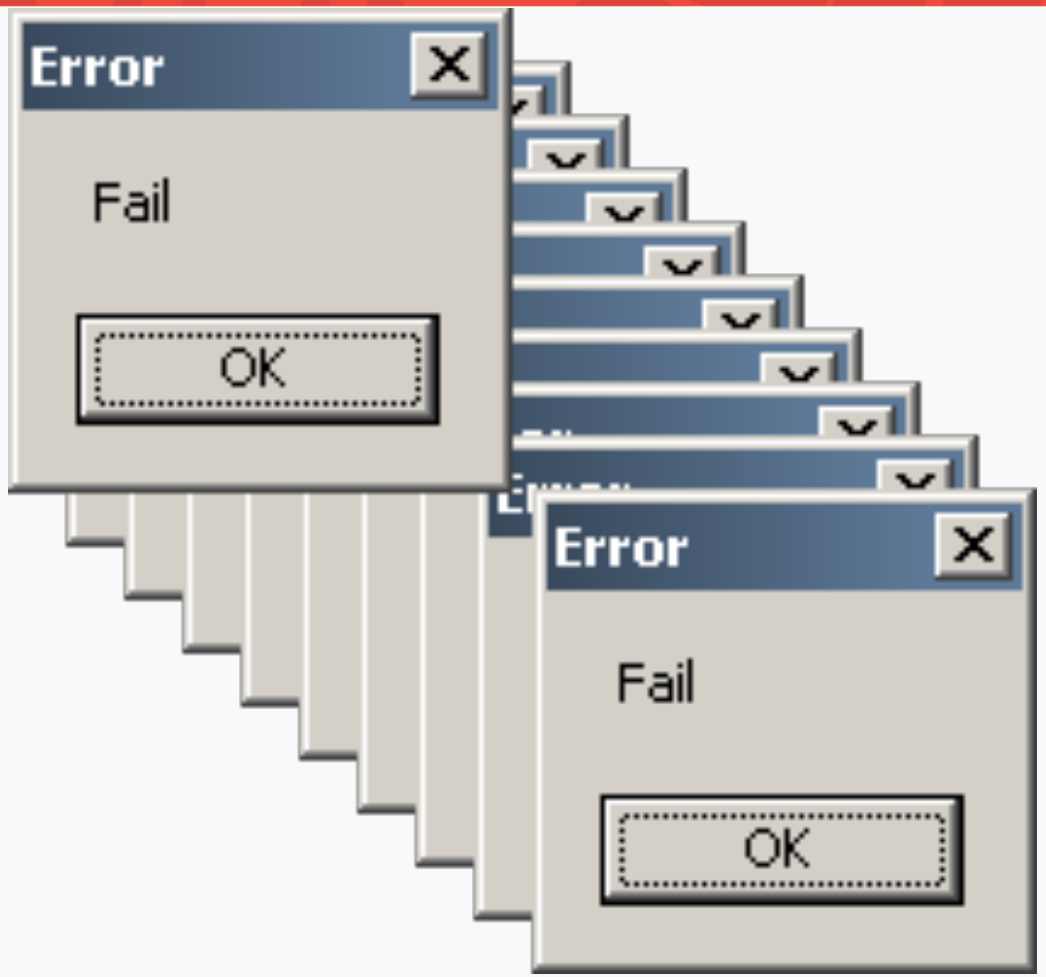
1630 Arctangent started.

1700 closed down.

Gen Knowledge

- BUG: Whenever a program/system is not behaving the way we expect
- Debugging is the process of figuring out the source of the error and fixing it.
- I think of it as the disconnect between my assumptions and what the code is actually doing.
- It's a skill, so you'll need to practice it. This is another great reason to help your peers.

Error Messages



Error Messages

The First rule of debugging:

- Read the error message!

Error Messages

The First rule of debugging:

- Read the error message!

The Second rule of debugging:

- Read the error message!

Error Messages

Errors are your friends!

- Analyze the message
- Note the line number(s)

- stack trace

Strategies

- Read the error messages!
- Know what your expected behavior is.
- Understand the input(s).
- Understand the program state.
- Make incremental changes.
- Check your assumptions.

Strategies

- Debug “inline”
- Use a REPL (irb || pry)
- Guess && Check
- Raise – Rescue

Debugging “inline”

- Use p statements to quickly show variable’s value

- Quickly determine if you are reaching a method
- Use “signaling code” to easily flag your spot
  ie: p “~” * 80

Tools

awesome_print

```
gem install awesome_print

require ‘awesome_print’
ap some_array
ap some_hash
```

PRY – The IRB Alternative

Pry is a REPL (Read-Eval-Print-Loop) much like IRB but with 3 additional key features:

- Syntax Highlighting
- Built in methods
- A Debugger
- Tabbed completion

PRY – Install

```
gem install pry
gem install pry-doc
gem install pry-byebug
rbenv rehash
```

PRY – terminal commands

ls (list methods)
_ (the last output)
? (show-doc)
. (send command to bash)
cat filename (displays the given file)
wtf? (wtf.....)

PRY#show-doc

```
[7] pry(main)> show-doc Array#each_with_index

From: enum.c (C Method):
Owner: Enumerable
Visibility: public
Signature: each_with_index(*arg1)
Number of lines: 11

Calls block with two arguments, the item and its index,
for each item in enum.  Given arguments are passed through
to #each().

If no block is given, an enumerator is returned instead.

   hash = Hash.new
   %w(cat dog wombat).each_with_index { |item, index|
     hash[item] = index
   }
   hash   #=> {"cat"=>0, "dog"=>1, "wombat"=>2}
[8] pry(main)>
```

pry-byebug commands

step: Step execution into the next line or method. Takes an optional numeric argument to step multiple times.

next: Step over to the next line within the same frame. Also takes an optional numeric argument to step multiple lines.

finish: Execute until current stack frame returns.

continue: Continue program execution and end the Pry session.

up: Moves the stack frame up. Takes an optional numeric argument to move multiple frames.

down: Moves the stack frame down. Takes an optional numeric argument to move multiple frames.

pry-byebug

gem install pry-byebug

require "pry-byebug”

binding.pry to stop execution and enter the REPL

Exceptions

- An instance of the Exception class
- A raised exception will propagate through each method in the call stack until it is stopped or reaches the point where the program started
- Raising and rescuing exceptions

Raise leads to Rescue

If we just had a raise with no conditions our code would never run.

Use a rescue to handle the error generated by raise and render user useful data back instead of a giant fail whale.

Rescue to the Rescue?

Careful:
Rescue isn't the bug free savior you might think

Why You Should Never Rescue Exception in Ruby

Debugging Wrapup

Questions?

Sweet Links

Pry Usage (youtube)

Replace IRB with PRY

Final Thought

Just saying; a well tested app will greatly reduce the amount of debugging you do each day….