Debugging

Spring 2019

Slides selected and slightly modified from Andreas Zeller, *Why Programs Fail*
https://www.st.cs.uni-saarland.de/whyprogramsfail/slides.php

(This work is licensed under the Creative Commons Attribution License. To view a copy of this license, visit http://creativecommons.org/licenses/by/1.0)
How Failures Come to be

Andreas Zeller
The First Bug
September 9, 1947

First actual case of bug being found.
Relay #70 Panel F
(moth) in relay.

1545
A Sample Program

$ sample 9 8 7
Output: 7 8 9

$ sample 11 14
Output: 0 11
How to Debug
(Sommerville 2004)

1. Locate error
2. Design error repair
3. Repair error
4. Re-test program
1. The programmer creates a defect – an error in the code.

2. When executed, the defect creates an infection – an error in the state.

3. The infection propagates.

4. The infection causes a failure.

This infection chain must be traced back – and broken.
The Curse of Testing

- Not every defect causes a failure!
- *Testing can only show the presence of errors – not their absence.*
  (Dijkstra 1972)
• Every failure can be traced back to some infection, and every infection is caused by some defect.

• Debugging means to relate a given failure to the defect – and to remove the defect.
Search in Space + Time

<table>
<thead>
<tr>
<th>✔</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

variables
The Defect

variables

Diagram showing a defect with a green checkmark and red Xs.
Search in Time

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
</tbody>
</table>

variables

t
Search in Time

variables

- ✓
- ✘
- ✓
- ✘
- ✓
- ✘
Search in Time

variables
Search in Space

variables
Search in Space

variables
Search in Space

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

variables
A Program State
A Sample Program

$ sample 9 8 7
Output: 7 8 9

$ sample 11 14
int main(int argc, char *argv[]) {
    int *a;
    int i;

    a = (int *)malloc((argc - 1) * sizeof(int));
    for (i = 0; i < argc - 1; i++)
        a[i] = atoi(argv[i + 1]);

    shell_sort(a, argc);

    printf("Output: ");
    for (i = 0; i < argc - 1; i++)
        printf("%d ", a[i]);
    printf("\n");

    free(a);

    return 0;
}
Find Origins

- The 0 printed is the value of \( a[0] \). Where does it come from?
- Basic idea: Track or deduce value origins
- Separates relevant from irrelevant values
- We can trace back \( a[0] \) to shell_sort
static void shell_sort(int a[], int size) {
    int i, j;
    int h = 1;
    do {
        h = h * 3 + 1;
    } while (h <= size);
    do {
        h /= 3;
        for (i = h; i < size; i++) {
            int v = a[i];
            for (j = i; j >= h && a[j - h] > v; j -= h) {
                a[j] = a[j - h];
            }
            if (i != j) {
                a[j] = v;
            }
        }
    } while (h != 1);
}
Observing a Run

Variables:
- `argv[0]` = 3
- `argv[1]` = "11" "14"
- `argv[2]` = "11" "14"
- `a[0]` = 0
- `a[1]` = 11
- `a[2]` = 14
- `i` = 0
- `size` = 3
- `h` = 2

1. `a = malloc(...)`
2. `i = 0`
3. `a[i] = atoi(argv[i + 1])`  
   - `a[0] = 11`
   - `i++`
4. `a[i] = atoi(argv[i + 1])`  
   - `a[1] = 11`
   - `i++`
5. `a[i] = atoi(argv[i + 1])`  
   - `a[2] = 14`
   - `i++`
6. `shell_sort(a, argc)`
7. `return 0`
Specific Observation

static void shell_sort(int a[], int size)
{
    fprintf(stderr, “At shell_sort”);
    for (i = 0; i < size; i++)
        fprintf(stderr, “a[%d] = %d\n”, i, a[i]);
    fprintf(stderr, “size = %d\n”, size);
    int i, j;
    int h = 1;
    ...
}

The state is infected at the call of shell_sort!
Fixing the Program

```c
int main(int argc, char *argv[])
{
    int *a;
    int i;

    a = (int *)malloc((argc - 1) * sizeof(int));
    for (i = 0; i < argc - 1; i++)
        a[i] = atoi(argv[i + 1]);

    shell_sort(a, argc); 1);

    ... $ sample 11 14
    Output: 11 14
}
```
Finding Causes

Infected state

Sane state

The difference causes the failure
Search in Space

Infected state

Sane state

Mixed state

Test

arginc = 3
Search in Time

Failing run

Passing run

\[ \text{argvc} = 3 \]

Transition from \text{argvc} to \text{a[2]}

\[ \text{a[2]} = 0 \]
int main(int argc, char *argv[]) {
    int *a;

    // Input array
    a = (int *)malloc((argc - 1) * sizeof(int));
    for (int i = 0; i < argc - 1; i++)
        a[i] = atoi(argv[i + 1]);

    // Sort array
    shell_sort(a, argc); // Should be argc - 1

    // Output array
    printf("Output: ");
    for (int i = 0; i < argc - 1; i++)
        printf("%d ", a[i]);
    printf("\n");

    free(a);
    return 0;
}