Amazing demos, everyone!

Take-aways ranged from:
• Really helpful to learn about more UI tooling, host tooling, test tooling AND authentication tooling options
• Appreciate that most everyone is going through similar learnings on the value of breaking down tasks, being agile with planning and designing (revisions happen!), learning and adapting to each others’ strengths in the group to
• Helped us identify some bugs
• Story telling of a demo is powerful
Today’s outline

• Debugging basics
• Delta debugging technique
  • In-class exercise on 11/15 will complement this material

Background Reading (link on the Calendar):
Simplifying and Isolating Failure-Inducing Input, Zeller and Hildebandt, 2002
Let’s level set

A bug is an **error or flaw** in a program or system that produces incorrect or **unexpected results**

Debugging is the process to **identify** and **resolve** bugs
The typical debugging process

- **Identify** – it’s a bug, not a feature
- **Understand** – what are the inputs and conditions causing the error
- **Reproduce** – create a (minimal) test to illustrate the issue
- **Investigate** – locate the problematic code
- **Fix** the code
- **Validate** and capture in a regression test
What’s a good bug (issue) report look like?

**A bug report** should be as specific as possible so that the engineer knows how to recreate the failure
- Provide information to reproduce the bug, including context
- What might be “context”?
- Also! Provide criticality of the bug (to influence priority in getting it fixed)

**A test case** should be as simple as possible
- Why?
Delta Debugging

A debugging technique to create a minimal test case that exposes the bug
This is a crashing test case

- Case crashed Mozilla
- Consider 370 of these being filed!
- What context is sufficient to expose the bug?
This is a crashing test case

- Crashed Mozilla
- What content is sufficient to expose the bug?
- A minimal test case is: `<SELECT>`
- Can we automate the process of minimizing test cases?
Minimizing test cases

Think of each test case as an input file with $n$ lines
Minimizing test cases

Test case

Failing

Test case

Passing

Test case

Passing
Minimizing test cases

Goal: minimize the failing test case
The happy path: binary search

Failing test with 16 lines
The minimal failing test has 2 lines: 3 and 4
The happy path: binary search
The happy path: binary search

<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>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>F</td>
<td></td>
<td>P</td>
<td></td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- F
- P
- P
- F
The happy path: binary search
The happy path: binary search

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

P F P

F
The happy path: binary search

Successfully minimized the failing test to 2 lines
The not so happy path...

Suppose the failure pattern is more complex
All three lines must exist in a failing test case: 3, 4, and 9
The not so happy path...

1          2            3           4             5            6            7            8            9           10         11 12          13         14          15        16

Binary search does not give optimal results
Delta Debugging = binary search + X

See paper – link on Calendar:
Simplifying and Isolating Failure-Inducing Input
Zeller and Hildebandt, 2002
The Delta Debugging algorithm

Four basic phases:
1. Test each subset
2. Test each complement
3. Increase granularity (increase # subsets)
4. Reduce

Complement example:
List: 1, 2, 3, 4
Complement of 1: 2, 3, 4

Minimizing Delta Debugging Algorithm

Let test and \( c_x \) be given such that \( test(\emptyset) = \checkmark \land test(c_x) = \times \) hold.
The goal is to find \( c'_x = \text{ddmin}(c_x) \) such that \( c'_x \subseteq c_x \), \( test(c'_x) = \times \), and \( c'_x \) is 1-minimal.
The minimizing Delta Debugging algorithm \( \text{ddmin}(c) \) is

\[
\text{ddmin}(c_x) = \text{ddmin}_2(c_x, 2) \quad \text{where}
\]

\[
\text{ddmin}_2(c'_x, n) = \begin{cases} 
\text{ddmin}_2(\Delta_i, 2) & \text{if } \exists i \in \{1, \ldots, n\} \cdot test(\Delta_i) = \times \text{ (“reduce to subset”)}
\\
\text{ddmin}_2(\bigvee_i \max(n - 1, 2)) & \text{else if } \exists i \in \{1, \ldots, n\} \cdot test(\bigvee_i) = \times \text{ (“reduce to complement”)}
\\
\text{ddmin}_2(c'_x, \min(|c'_x|, 2n)) & \text{else if } n < |c'_x| \text{ (“increase granularity”)}
\\
c'_x & \text{otherwise (“done”).}
\end{cases}
\]

where \( \bigvee_i = c'_x - \Delta_i, c'_x = \Delta_1 \cup \Delta_2 \cup \cdots \cup \Delta_n \), all \( \Delta_i \) are pairwise disjoint, and \( \forall \Delta_i : |\Delta_i| \approx |c'_x|/n \) holds.
The recursion invariant (and thus precondition) for \( \text{ddmin}_2 \) is \( test(c'_x) = \times \land n \leq |c'_x| \).
Delta Debugging: it’s mostly binary search
Delta Debugging: test subsets

\[ \Delta_1 \]

\[ \Delta_2 \]

Notation for subset

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

P P

F F
Delta Debugging: increase granularity

\[ \Delta_1, \Delta_2, \Delta_3, \Delta_4 \]

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
F \quad F
\]
Delta Debugging: complements

The order in which deltas and complements are evaluated may differ between implementations of the algorithm.

Notation for complement of subset 1
Delta Debugging: complements

\[ \nabla_4 \]

\[ \Delta_4 \]

F

F

F

P

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Delta Debugging: reduce

\[ \Delta_1 \quad \Delta_2 \quad \Delta_3 \]

1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16

P   P   P   P

F   F   F
Delta Debugging: complements
Delta Debugging: reduce
Delta Debugging: increase granularity
Delta Debugging: complements

And so on...
Delta Debugging: finds a 1-minimal solution

Failing test cases must be deterministic and monotone
Delta debugging: live example
Delta Debugging: live example

Program and initial test case
- Program P crashes whenever the input contains 1 2 8
- Initial crashing test case is: 1 2 3 4 5 6 7 8

Syntax:
- % ./delta -test=./test.sh -cp_minimal=./min.txt < failing.txt
- test.sh returns 0 if input causes failure, 1 if input passes

Delta debugging approach:
- Test each subset
- Test each complement
- Increase granularity
- Reduce

Example taken from the reading
Delta debugging: one more example
Let’s try one more

Program and initial test case
- Program $P$ takes as input a list of integers $l$.
- $P$ crashes whenever $l$ contains 4,2.
- Initial crashing test case is: 2,4,2,4

Complete the following table

<table>
<thead>
<tr>
<th>Iteration</th>
<th>n</th>
<th>input</th>
<th>$\Delta_1, \ldots, \Delta_n$</th>
<th>$\nabla_1, \ldots, \nabla_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2424</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Let’s try one more

Program and initial test case
- Program $P$ takes as input a list of integers $l$.
- $P$ crashes whenever $l$ contains 4,2.
- Initial crashing test case is: 2,4,2,4

Complete the following table

<table>
<thead>
<tr>
<th>Iteration</th>
<th>n</th>
<th>input</th>
<th>$\Delta_1, \ldots, \Delta_n$</th>
<th>$\nabla_1, \ldots, \nabla_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2424</td>
<td>24, (24)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>2424</td>
<td>2, 4, (2), (4), 424, (224), (244), (242)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>424</td>
<td>(4), (2), (4), (24), 44, 42</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>42</td>
<td>(4), (2)</td>
<td></td>
</tr>
</tbody>
</table>
Delta debugging: in-class exercise Wednesday
Peer review assignment posted

Reading #3 posted - no summary paragraph required