Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

Event-based application debugging

Alexander Elyasov, Wishnu Prasetya, Jurriaan Hage

Department of Information and Computing Sciences, Universiteit Utrecht
A.Elyasov@uu.nl

August 21, 2013

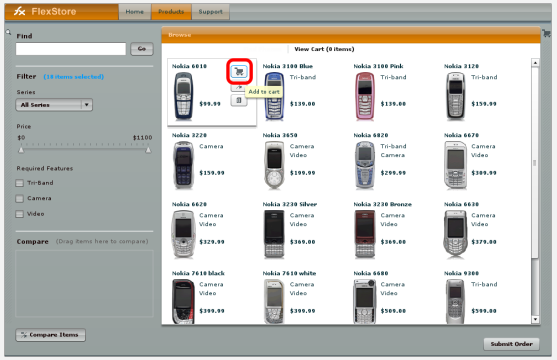
FITTEST

Failure Scenario

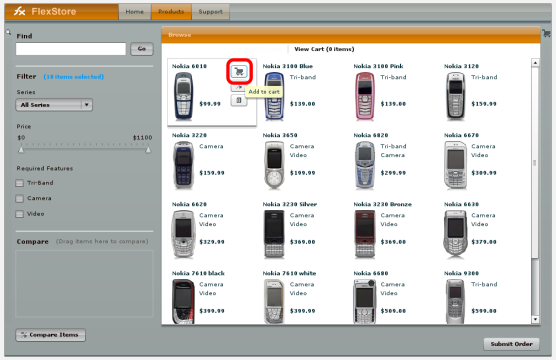
Universiteit Utrecht

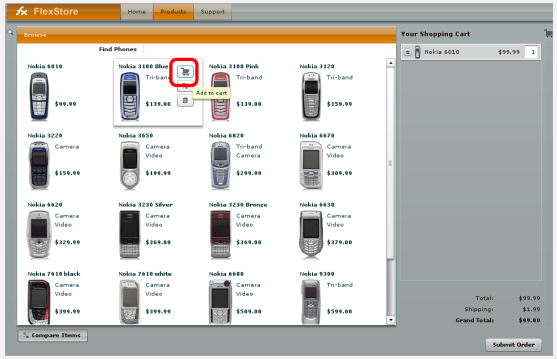
[Faculty of **Science**
Information and Computing Sciences]

Failure Scenario

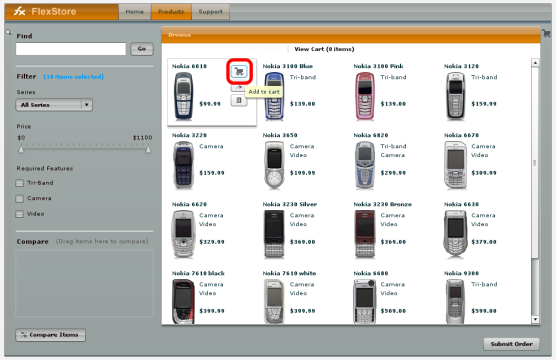


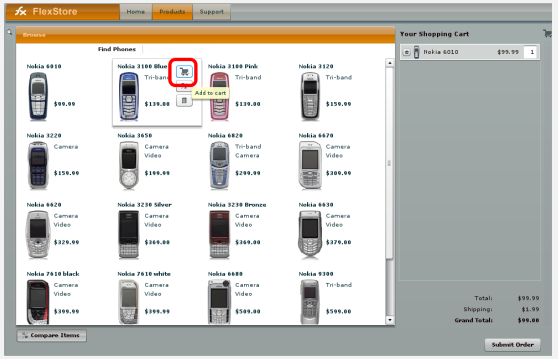
Failure Scenario

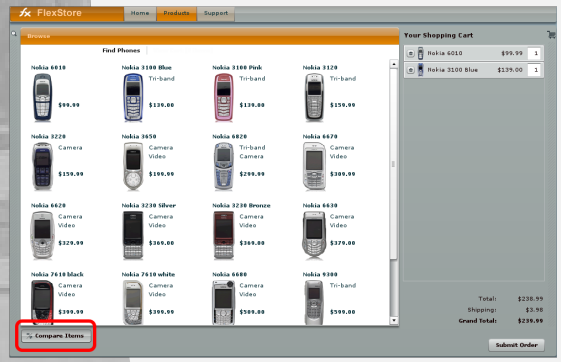




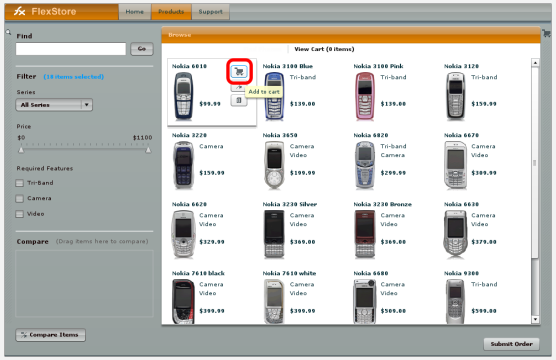
Failure Scenario

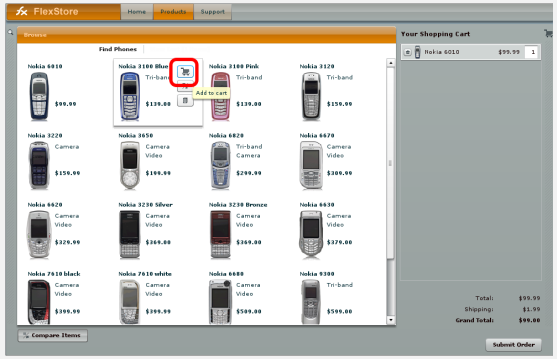


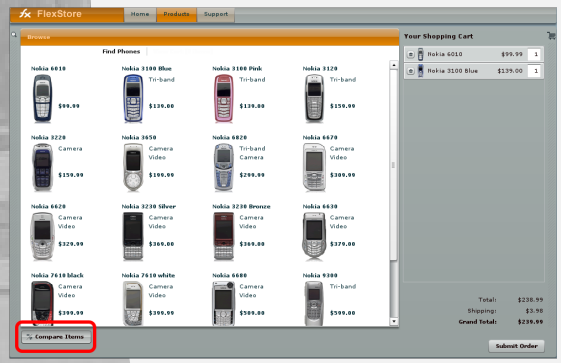


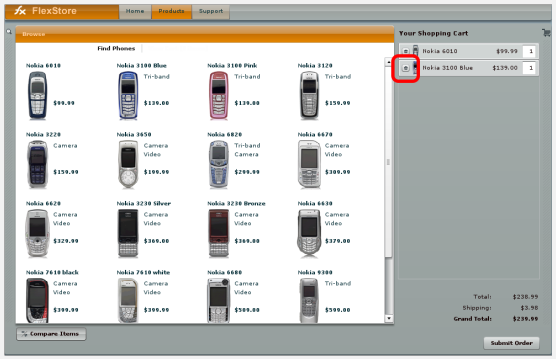


Failure Scenario

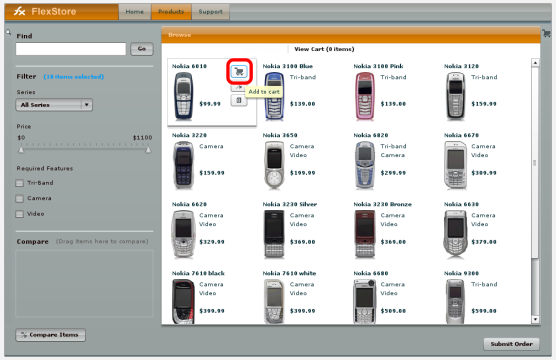


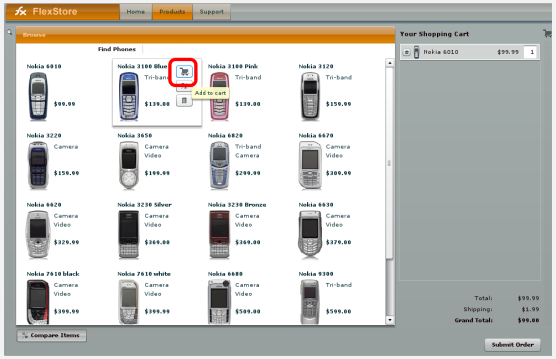


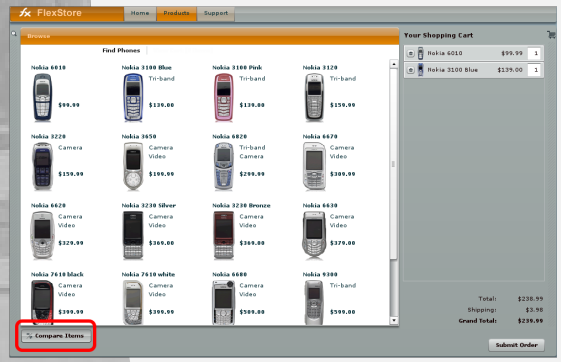


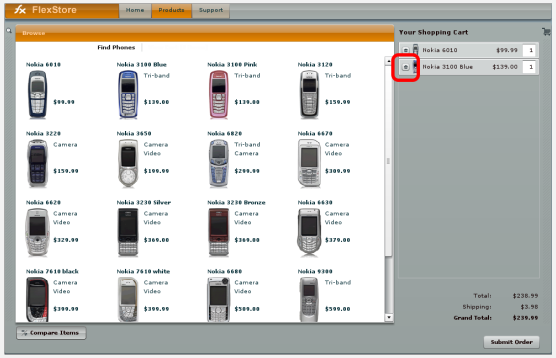


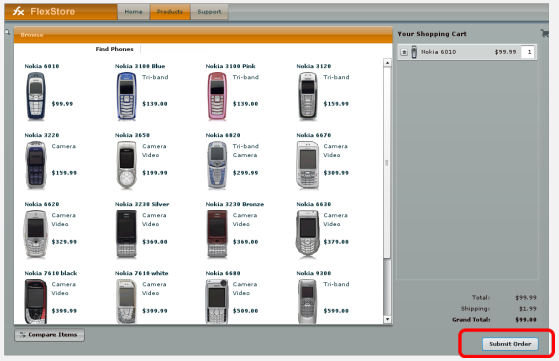
Failure Scenario



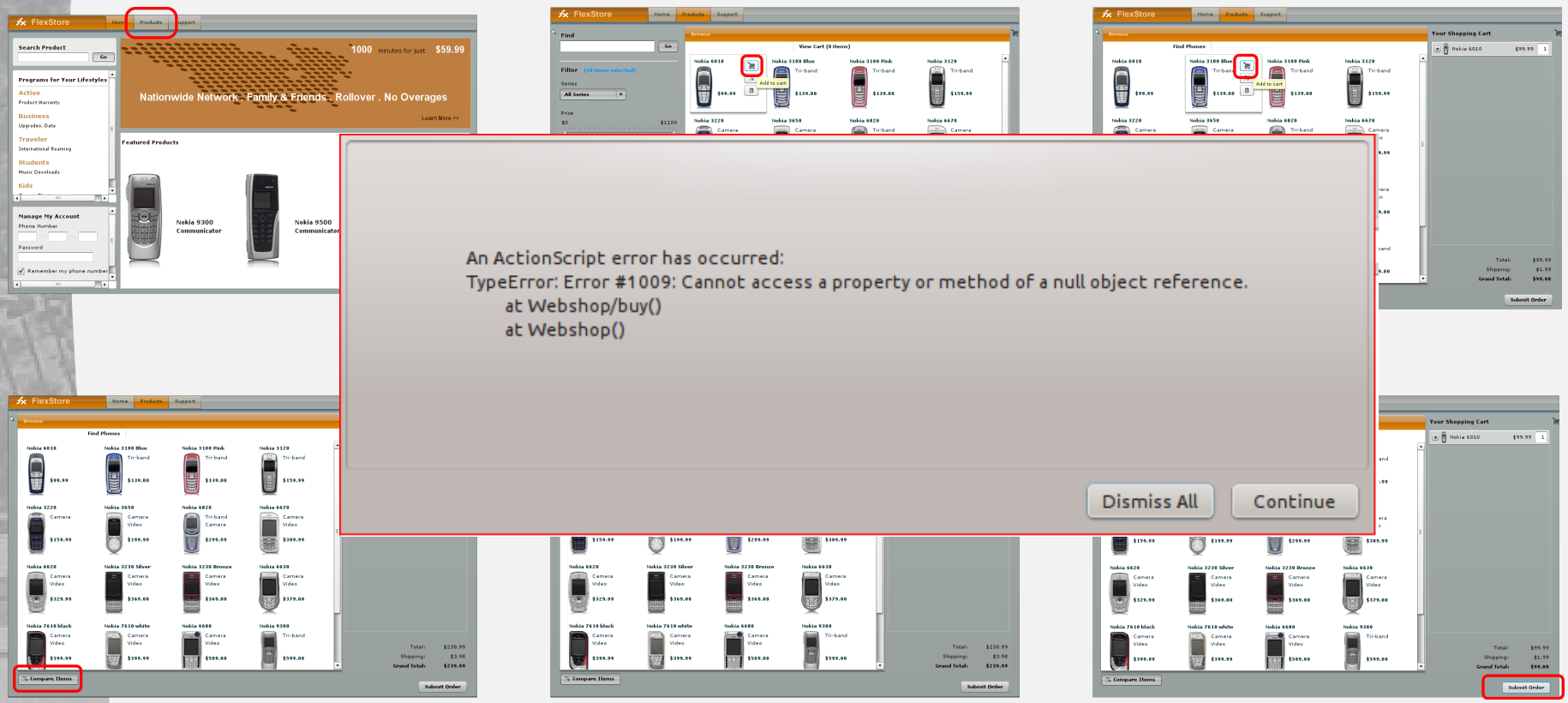








Failure Scenario



Error in the Source Code

```
1       public class PhoneShop {
          public var cart:Array; // list of phone id-s in the cart
3         public var catalog:Array;  // list of all products {id, price}
          public function PhoneShop():void {
5           catalog = getCatalogFromXML("./catalog.xml");
            // catalog.xml contains 18 records
7         }
          public function add(id:int):void {
9           cart.add(18); /** instead of cart.add(id) !!!*/
          }
11        public function buy():int {
            var total:int = 0;
13          for (var i:int = 0; i < cart.length; i++) {
              total += catalog.find(cart[i]).price;
15          }
            return total;
17        }
          public function delete(id):void {...}
19        ...
        }
```

Minimizing the test case with Delta Debugging

#	n	Δ	Steps						R	N
0	$n = 1$	Δ_0	open	add(1)	add(2)	com	del(2)	buy	✘	

Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

Minimizing the test case with Delta Debugging

#	n	Δ	Steps						R	N
0	$n = 1$	Δ_0	open	add(1)	add(2)	com	del(2)	buy	✘	
1	$n = 2$	Δ_1	-	-	-	com	del(2)	buy	?	
2	$n = 2$	∇_1	open	add(1)	add(2)	-	-	-	✔	

Minimizing the test case with Delta Debugging

#	n	Δ	Steps						R	N
0	$n = 1$	Δ_0	open	add(1)	add(2)	com	del(2)	buy	✘	
1	$n = 2$	Δ_1	-	-	-	com	del(2)	buy	?	
2	$n = 2$	∇_1	open	add(1)	add(2)	-	-	-	✔	
3	$n = 4$	Δ_1	-	add(1)	add(2)	com	del(2)	buy	?	
4	$n = 4$	Δ_2	open	-	add(2)	com	del(2)	buy	?	
5	$n = 4$	Δ_3	open	add(1)	-	-	del(2)	buy	?	
6	$n = 4$	Δ_4	open	add(1)	add(2)	com	-	-	✔	
7	$n = 4$	∇_1	open	-	-	-	-	-	✔	
8	$n = 4$	∇_2	-	add(1)	-	-	-	-	?	
9	$n = 4$	∇_3	-	-	add(2)	com	-	-	?	
10	$n = 4$	∇_4	-	-	-	-	del(2)	buy	?	

Minimizing the test case with Delta Debugging

#	n	Δ	Steps						R	N
0	$n = 1$	Δ_0	open	add(1)	add(2)	com	del(2)	buy	✘	
1	$n = 2$	Δ_1	-	-	-	com	del(2)	buy	?	
2	$n = 2$	∇_1	open	add(1)	add(2)	-	-	-	✔	
3	$n = 4$	Δ_1	-	add(1)	add(2)	com	del(2)	buy	?	
4	$n = 4$	Δ_2	open	-	add(2)	com	del(2)	buy	?	
5	$n = 4$	Δ_3	open	add(1)	-	-	del(2)	buy	?	
6	$n = 4$	Δ_4	open	add(1)	add(2)	com	-	-	✔	
7	$n = 4$	∇_1	open	-	-	-	-	-	✔	
8	$n = 4$	∇_2	-	add(1)	-	-	-	-	?	
9	$n = 4$	∇_3	-	-	add(2)	com	-	-	?	
10	$n = 4$	∇_4	-	-	-	-	del(2)	buy	?	
11	$n = 6$	Δ_1	-	add(1)	add(2)	com	del(2)	buy	?	3
12	$n = 6$	Δ_2	open	-	add(2)	com	del(2)	buy	✔	4
13	$n = 6$	Δ_3	open	add(1)	-	com	del(2)	buy	?	
14	$n = 6$	Δ_4	open	add(1)	add(2)	-	del(2)	buy	✘	

Minimizing the test case with Delta Debugging

#	n	Δ	Steps						R	N
0	$n = 1$	Δ_0	open	add(1)	add(2)	com	del(2)	buy	✘	
1	$n = 2$	Δ_1	-	-	-	com	del(2)	buy	?	
2	$n = 2$	∇_1	open	add(1)	add(2)	-	-	-	✔	
3	$n = 4$	Δ_1	-	add(1)	add(2)	com	del(2)	buy	?	
4	$n = 4$	Δ_2	open	-	add(2)	com	del(2)	buy	?	
5	$n = 4$	Δ_3	open	add(1)	-	-	del(2)	buy	?	
6	$n = 4$	Δ_4	open	add(1)	add(2)	com	-	-	✔	
7	$n = 4$	∇_1	open	-	-	-	-	-	✔	
8	$n = 4$	∇_2	-	add(1)	-	-	-	-	?	
9	$n = 4$	∇_3	-	-	add(2)	com	-	-	?	
10	$n = 4$	∇_4	-	-	-	-	del(2)	buy	?	
11	$n = 6$	Δ_1	-	add(1)	add(2)	com	del(2)	buy	?	3
12	$n = 6$	Δ_2	open	-	add(2)	com	del(2)	buy	✔	4
13	$n = 6$	Δ_3	open	add(1)	-	com	del(2)	buy	?	
14	$n = 6$	Δ_4	open	add(1)	add(2)	-	del(2)	buy	✘	
15	$n = 2$	Δ_1	-	-	add(2)	-	del(2)	buy	?	
16	$n = 2$	∇_1	open	add(1)	-	-	-	-	✔	

Minimizing the test case with Delta Debugging

#	n	Δ	Steps						R	N
0	$n=1$	Δ_0	open	add(1)	add(2)	com	del(2)	buy	✘	
1	$n=2$	Δ_1	-	-	-	com	del(2)	buy	?	
2	$n=2$	∇_1	open	add(1)	add(2)	-	-	-	✔	
3	$n=4$	Δ_1	-	add(1)	add(2)	com	del(2)	buy	?	
4	$n=4$	Δ_2	open	-	add(2)	com	del(2)	buy	?	
5	$n=4$	Δ_3	open	add(1)	-	-	del(2)	buy	?	
6	$n=4$	Δ_4	open	add(1)	add(2)	com	-	-	✔	
7	$n=4$	∇_1	open	-	-	-	-	-	✔	
8	$n=4$	∇_2	-	add(1)	-	-	-	-	?	
9	$n=4$	∇_3	-	-	add(2)	com	-	-	?	
10	$n=4$	∇_4	-	-	-	-	del(2)	buy	?	
11	$n=6$	Δ_1	-	add(1)	add(2)	com	del(2)	buy	?	3
12	$n=6$	Δ_2	open	-	add(2)	com	del(2)	buy	✔	4
13	$n=6$	Δ_3	open	add(1)	-	com	del(2)	buy	?	
14	$n=6$	Δ_4	open	add(1)	add(2)	-	del(2)	buy	✘	
15	$n=2$	Δ_1	-	-	add(2)	-	del(2)	buy	?	
16	$n=2$	∇_1	open	add(1)	-	-	-	-	✔	
17	$n=4$	Δ_1	-	add(1)	add(2)	-	del(2)	buy	?	
18	$n=4$	Δ_2	open	-	add(2)	-	del(2)	buy	✔	
19	$n=4$	Δ_3	open	add(1)	-	-	del(2)	buy	?	
20	$n=4$	Δ_4	open	add(1)	add(2)	-	-	buy	✘	

Minimizing the test case with Delta Debugging

#	n	Δ	Steps						R	N
21	$n = 2$	Δ_1	-	-	add(2)	-	-	buy	?	
22	$n = 2$	∇_1	open	add(1)	-	-	-	-	✔	19

Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

Minimizing the test case with Delta Debugging

#	n	Δ	Steps						R	N
21	$n = 2$	Δ_1	-	-	add(2)	-	-	buy	?	
22	$n = 2$	∇_1	open	add(1)	-	-	-	-	✔	19
23	$n = 4$	Δ_1	-	add(1)	add(2)	-	-	buy	?	
24	$n = 4$	Δ_2	open	-	add(2)	-	-	buy	✘	

Minimizing the test case with Delta Debugging

#	n	Δ	Steps						R	N
21	$n = 2$	Δ_1	-	-	add(2)	-	-	buy	?	
22	$n = 2$	∇_1	open	add(1)	-	-	-	-	✔	19
23	$n = 4$	Δ_1	-	add(1)	add(2)	-	-	buy	?	
24	$n = 4$	Δ_2	open	-	add(2)	-	-	buy	✘	
25	$n = 2$	Δ_1	-	-	add(2)	-	-	buy	?	
26	$n = 2$	∇_1	open	-	-	-	-	-	✔	7

Minimizing the test case with Delta Debugging

#	n	Δ	Steps						R	N
21	$n = 2$	Δ_1	-	-	add(2)	-	-	buy	?	
22	$n = 2$	∇_1	open	add(1)	-	-	-	-	✔	19
23	$n = 4$	Δ_1	-	add(1)	add(2)	-	-	buy	?	
24	$n = 4$	Δ_2	open	-	add(2)	-	-	buy	✘	
25	$n = 2$	Δ_1	-	-	add(2)	-	-	buy	?	
26	$n = 2$	∇_1	open	-	-	-	-	-	✔	7
27	$n = 3$	Δ_1	-	-	add(2)	-	-	buy	?	21
28	$n = 3$	Δ_2	open	-	-	-	-	buy	✔	
29	$n = 3$	Δ_3	open	-	add(2)	-	-	-	✔	
30	$n = 3$	∇_1	open	-	-	-	-	-	✔	7
31	$n = 3$	∇_1	-	-	add(2)	-	-	-	?	
32	$n = 3$	∇_1	-	-	-	-	-	buy	?	

total: 26 test runs

Properties of Delta Debugging

- Complexity:
  $2\log_2(|c_{\text{✘}}|) \leqslant |\text{number of tests}| \leqslant |c_{\text{✘}}|^2 + 3|c_{\text{✘}}|$.
- Simplification (ddmin) removes any single event that would cause the failure to disappear.
- Isolation (dd) removes a particular event (set) that would cause the failure to disappear.
- dd is a generalization of ddmin which is in general more efficient.
- However, in this work we focus on ddmin.

Problems of Delta Debugging

Problem

The length of the failing test case is often too large for effective search by DD.

Problems of Delta Debugging

Can we do something about it?

Problem

The length of the failing test case is often too large for effective search by DD.

Problems of Delta Debugging

Problem

The length of the failing test case is often too large for effective search by DD.

Yes, we can! Let's try to reduce the test case by identifying the set of suspicious events

Delta Debugging on the reduced test case

#	n	Δ	Steps						R	N
0	$n = 1$	Δ_1	open	add(1)	add(2)	com	del(2)	buy	✘	

Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

Delta Debugging on the reduced test case

#	n	Δ	Steps						R	N
0	$n = 1$	Δ_1	open	add(1)	add(2)	com	del(2)	buy	✘	
1	$n = 2$	Δ_1	-	add(1)	-	com	del(2)	buy	?	
2	$n = 2$	∇_1	open	add(1)	add(2)	-	-	buy	✘	

Delta Debugging on the reduced test case

#	n	Δ	Steps						R	N
0	$n = 1$	Δ_1	open	add(1)	add(2)	com	del(2)	buy	✘	
1	$n = 2$	Δ_1	-	add(1)	-	com	del(2)	buy	?	
2	$n = 2$	∇_1	open	add(1)	add(2)	-	-	buy	✘	
3	$n = 2$	Δ_1	-	-	add(2)	-	-	buy	?	
4	$n = 2$	∇_1	open	add(1)	-	-	-	-	✔	

Delta Debugging on the reduced test case

#	n	Δ	Steps						R	N
0	$n = 1$	Δ_1	open	add(1)	add(2)	com	del(2)	buy	✘	
1	$n = 2$	Δ_1	-	add(1)	-	com	del(2)	buy	?	
2	$n = 2$	∇_1	open	add(1)	add(2)	-	-	buy	✘	
3	$n = 2$	Δ_1	-	-	add(2)	-	-	buy	?	
4	$n = 2$	∇_1	open	add(1)	-	-	-	-	✔	
5	$n = 4$	Δ_1	-	add(1)	add(2)	-	-	buy	?	
6	$n = 4$	Δ_2	open	-	add(2)	-	-	buy	✘	

Delta Debugging on the reduced test case

#	n	Δ	Steps						R	N
0	$n = 1$	Δ_1	open	add(1)	add(2)	com	del(2)	buy	✘	
1	$n = 2$	Δ_1	-	add(1)	-	com	del(2)	buy	?	
2	$n = 2$	∇_1	open	add(1)	add(2)	-	-	buy	✘	
3	$n = 2$	Δ_1	-	-	add(2)	-	-	buy	?	
4	$n = 2$	∇_1	open	add(1)	-	-	-	-	✔	
5	$n = 4$	Δ_1	-	add(1)	add(2)	-	-	buy	?	
6	$n = 4$	Δ_2	open	-	add(2)	-	-	buy	✘	
7	$n = 2$	Δ_1	-	-	add(2)	-	-	buy	?	3
8	$n = 2$	∇_1	open	-	-	-	-	-	✔	

Delta Debugging on the reduced test case

#	n	Δ	Steps						R	N
0	$n = 1$	Δ_1	open	add(1)	add(2)	com	del(2)	buy	✘	
1	$n = 2$	Δ_1	-	add(1)	-	com	del(2)	buy	?	
2	$n = 2$	∇_1	open	add(1)	add(2)	-	-	buy	✘	
3	$n = 2$	Δ_1	-	-	add(2)	-	-	buy	?	
4	$n = 2$	∇_1	open	add(1)	-	-	-	-	✔	
5	$n = 4$	Δ_1	-	add(1)	add(2)	-	-	buy	?	
6	$n = 4$	Δ_2	open	-	add(2)	-	-	buy	✘	
7	$n = 2$	Δ_1	-	-	add(2)	-	-	buy	?	3
8	$n = 2$	∇_1	open	-	-	-	-	-	✔	
9	$n = 3$	Δ_1	-	-	add(2)	-	-	buy	?	3
10	$n = 3$	Δ_2	open	-	-	-	-	buy	✔	8
11	$n = 3$	Δ_3	open	-	add(2)	-	-	-	✔	
12	$n = 3$	∇_1	open	-	-	-	-	-	✔	8
13	$n = 3$	∇_1	-	-	add(2)	-	-	-	?	
14	$n = 3$	∇_1	-	-	-	-	-	buy	?	

total: 10 test runs

- **10** test runs vs. **26** test runs

Log-Based Reduction

How to identify the set of suspicious events?

- Define application abstract state
- Log this state together with application events
- Define equivalence relation on the logs
- Apply reduction procedure to the failing test case

Application Abstract State (AS)

- AS is an abstraction of the application state which is provided by the developer.
- It can be, for example, constructed from the variables participating in the oracles or assertions.
- In our web shop example, it makes sense to define AS as the list of phone ids in the `cart` variable.
- The quality of abstraction correlates with the number of tests carried out by the improved version of DD

Improved Logging Model

- Events may accept parameters
- Additionally, an application abstract state is logged
- A log is an alternating sequence of events and states:

$$\Sigma = [(\epsilon, s_0), (e_1, s_1), \ldots, (e_n, s_n)] = s_0 \rightarrow \tau,$$

where $\tau = [e_1, \ldots, e_n]$ and ϵ doesn't have any effect on the abstract state.

Equivalence relation on logs

- Two event sequences τ_1 and τ_2 are equivalent $(\tau_1 \equiv \tau_2)$ iff $\forall s : \Sigma_1 = s \rightarrow \tau_1 \wedge \Sigma_2 = s \rightarrow \tau_2 \Rightarrow lastSt(\Sigma_1) = lastSt(\Sigma_2)$
  - corresponding logs Σ_1 and Σ_2 are also equivalent $(\Sigma_1 \equiv \Sigma_2)$
- Given an event alphabet E, a log-reduction system is a tuple $(\Sigma(E^*), \rightarrow_R)$, where
  - $\Sigma(E^*)$ is a set of logs generated by the event sequences from E^*
  - $\rightarrow_R$ is a reduction relation imposed by the set of event equivalences R

Rewrite Rules

Rewrite Rule Patterns

$$\left.\begin{array}{ll} SKIP(e) & [e(p)] \equiv \epsilon \\ ZERO(d,e) & [d(p);e(q)] \equiv [e(q)] \\ COM(e,d) & [e(p);d(q)] \equiv [d(q);e(p)] \\ INV(e,d) & [e(p);x^*;d(p)] \equiv \epsilon \end{array}\right\} \Rightarrow R = \langle \Sigma(E^*), \rightarrow_R \rangle$$

Concrete Rewrite Rules

- $[open] = \epsilon$
- $[com] = \epsilon$
- $[add(i);x^*;del(i)] = \epsilon$

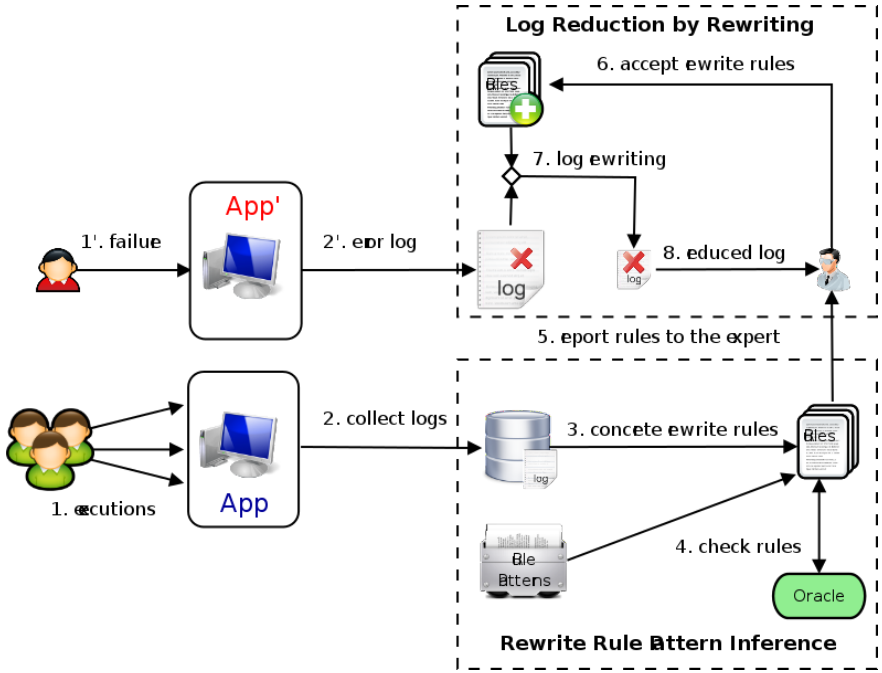
Log Reduction

$$
\begin{array}{ll}
 & [\ \textbf{open}\ ;\ add(1)\ ;\ add(2)\ ;\ com\ ;\ del(2)\ ;\ buy\ ] \\
\equiv & \{\ \text{apply [open]}= \epsilon\ \text{-}\} \\
 & [\ \ ;\ add(1)\ ;\ add(2)\ ;\ \textbf{com}\ ;\ del(2)\ ;\ buy\ ] \\
\equiv & \{\ \text{apply [com]}= \epsilon\ \text{-}\} \\
 & [\ \ ;\ add(1)\ ;\ \textbf{add(2)}\ ;\ \ ;\ \textbf{del(2)}\ ;\ buy\ ] \\
\equiv & \{\ \text{apply [add(2);del(2)]}= \epsilon\ \text{-}\} \\
 & [\ \ ;\ add(1)\ ;\ \ ;\ \ ;\ \ ;\ buy\ ]
\end{array}
$$

Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

Log Reduction Framework



Conclusion

- Initial search space for Delta Debugging can be shrunk by identifying the set of events likely having contribution to the failure.
- The set of suspicious events is obtained by applying reduction procedure to the failing sequence.
- Reduction system is based on the rewriting rules inferred from the collection of logs produced by the application during successful runs.
- The approach is sound even if the set of suspicious events doesn't make sense. Then, DD can be applied to the original sequence.

References

- A. Zeller and R. Hildebrandt, "Simplifying and isolating failure-inducing input," Software Engineering, IEEE Transactions on, vol. 28, no. 2, pp. 183–200, 2002.
- W. Prasetya, A. Elyasov, A. Middelkoop, and J. Hage, "Fittest log format (version 1.1)," Department of Information and Computing Sciences, Utrecht University, Tech. Rep. UU-CS-2012-014, 2012.
- A. Elyasov, W. Prasetya, and J. Hage, "Log-based reduction by rewriting," Department of Information and Computing Sciences, Utrecht University, Tech. Rep. UU-CS-2012-013, 2012.

The End

Thanks for attention!

Questions?