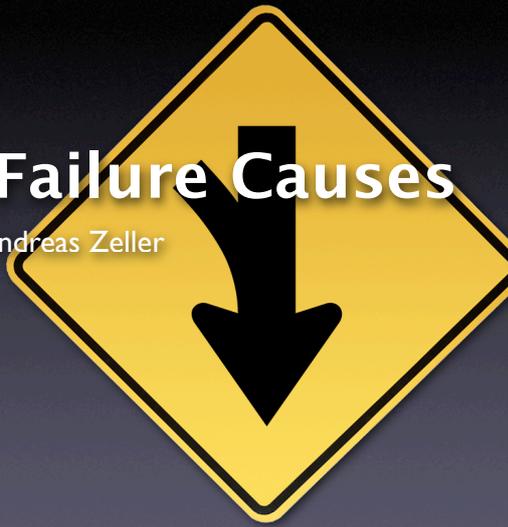


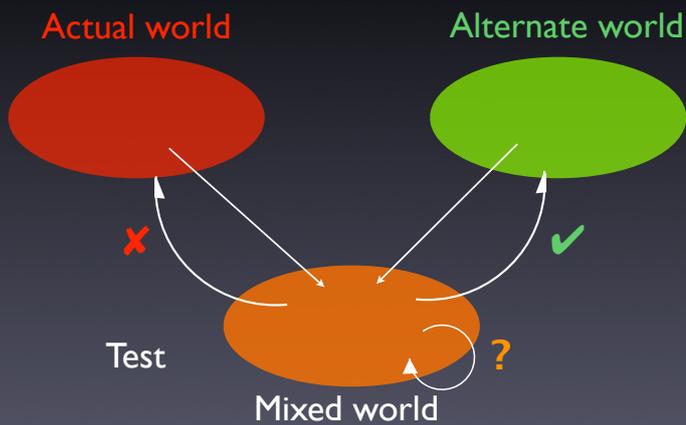
Isolating Failure Causes

Andreas Zeller



1

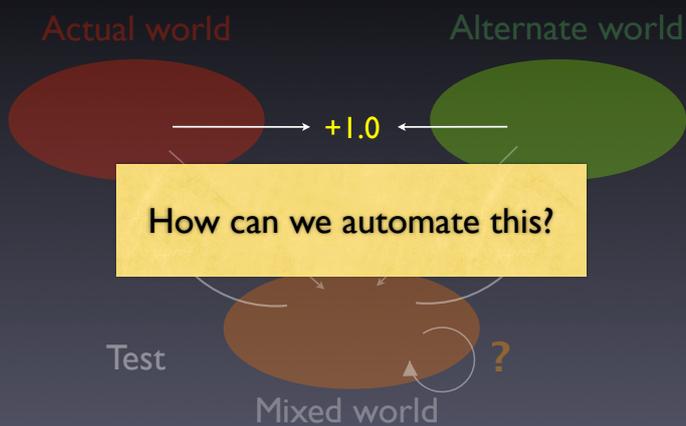
Isolating Causes



2

2

Isolating Causes



3

3

Simplifying Input

- <SELECT NAME="priority" MULTIPLE SIZE=7> ❌
- <SELECT NAME="priority" MULTIPLE SIZE=7> ✅
- <SELECT NAME="priority" MULTIPLE SIZE=7> ✅
- <SELECT NAME="priority" MULTIPLE SIZE=7> ✅
- <SELECT NAME="priority" MULTIPLE SIZE=7> ❌
- <SELECT NAME="priority" MULTIPLE SIZE=7> ❌
- <SELECT NAME="priority" MULTIPLE SIZE=7> ✅



Simplifying

Input

- [Redacted] ❌
- [Redacted] ❌
- [Redacted] ❌
- ⋮
- [Redacted] (Failure Cause) ❌
- [Redacted] ✅



Isolating Input

- <SELECT NAME="priority" MULTIPLE SIZE=7> ❌
- <SELECT NAME="priority" MULTIPLE SIZE=7> ✅
- <SELECT NAME="priority" MULTIPLE SIZE=7> ✅



Difference narrowed down

Isolating Input

<SELECT NAME="priority" MULTIPLE SIZE=7> ✗
 <SELECT NAME="priority" MULTIPLE SIZE=7> ✗
 <SELECT NAME="priority" MULTIPLE SIZE=7> ✓
 <SELECT NAME="priority" MULTIPLE SIZE=7> ✓

Failure Cause

7

7

Isolating

Input

Failure Cause

8

8

Finding Causes

Simplifying

Input

Failure Cause

6

Isolating

Input

Failure Cause

7

- minimal input
- minimal context

- minimal difference
- common context

9

9

Configuration

Circumstance

δ

All circumstances

$$C = \{\delta_1, \delta_2, \dots\}$$

Configuration $c \subseteq C$

$$c = \{\delta_1, \delta_2, \dots, \delta_n\}$$

10

10

Tests

Testing function

$$test(c) \in \{\checkmark, \times, ?\}$$

Initial configurations

$$test(c_{\checkmark}) = \checkmark$$

$$test(c_{\times}) = \times$$

11

11

Minimal Difference

Goal: Subsets c'_{\times} and c'_{\checkmark}

$$\emptyset = c_{\checkmark} \subseteq c'_{\checkmark} \subset c'_{\times} \subseteq c_{\times}$$

Difference

$$\Delta = c'_{\times} \setminus c'_{\checkmark}$$

Difference is 1-minimal

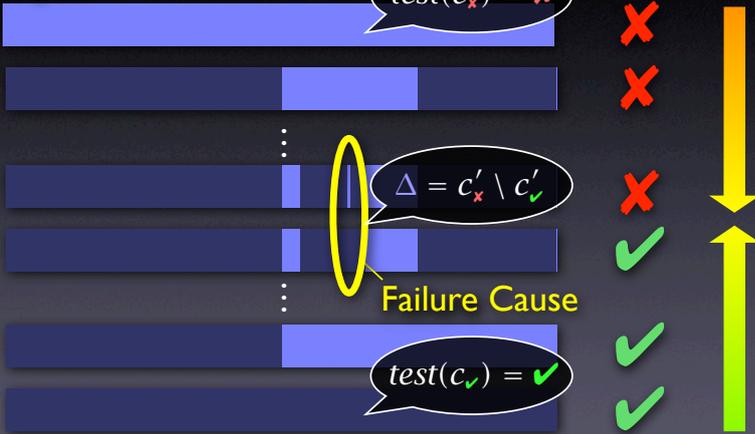
$$\forall \delta_i \in \Delta \cdot test(c'_{\checkmark} \cup \{\delta_i\}) \neq \checkmark \wedge test(c'_{\times} \setminus \{\delta_i\}) \neq \times$$

12

12

Isolating

Input



13

Algorithm Sketch

- Extend *ddmin* such that it works on *two sets at a time* – c'_x and c'_v
- Compute subsets

$$\Delta_1 \cup \Delta_2 \cup \dots \cup \Delta_n = \Delta = c'_x \setminus c'_v$$
- For each subset, test
 - the *addition* $c'_v \cup \Delta_i$
 - the *removal* $c'_x \setminus \Delta_i$

14

14

Test Outcomes

	\times	\checkmark
$(c'_x \setminus \Delta) = c'_x \setminus \Delta$	$(c'_x \setminus \Delta) = c'_x \setminus \Delta$	$(c'_x \setminus \Delta) = c'_x \setminus \Delta$
$(c'_v \cup \Delta) = c'_v \cup \Delta$	$(c'_v \cup \Delta) = c'_v \cup \Delta$	$(c'_v \cup \Delta) = c'_v \cup \Delta$
otherwise	increase granularity	

most valuable outcomes

15

15

dd in a Nutshell

$dd(c_{\checkmark}, c_{\times}) = (c'_{\checkmark}, c'_{\times})$ $\Delta = c'_{\times} \setminus c'_{\checkmark}$ is 1-minimal

$dd(c_{\checkmark}, c_{\times}) = dd'(c_{\checkmark}, c_{\times}, 2)$

$dd'(c'_{\checkmark}, c'_{\times}, n) =$

$(c'_{\checkmark}, c'_{\times})$	if $ \Delta = 1$
$dd'(c'_{\times} \setminus \Delta_i, c'_{\times}, 2)$	if $\exists i \in \{1..n\} \cdot test(c'_{\times} \setminus \Delta_i) = \checkmark$
$dd'(c'_{\checkmark}, c'_{\checkmark} \cup \Delta_i, 2)$	if $\exists i \in \{1..n\} \cdot test(c'_{\checkmark} \cup \Delta_i) = \times$
$dd'(c'_{\checkmark} \cup \Delta_i, c'_{\times}, \max(n-1, 2))$	else if $\exists i \in \{1..n\} \cdot test(c'_{\checkmark} \cup \Delta_i) = \checkmark$
$dd'(c'_{\times}, c'_{\times} \setminus \Delta_i, \max(n-1, 2))$	else if $\exists i \in \{1..n\} \cdot test(c'_{\times} \setminus \Delta_i) = \times$
$dd'(c'_{\checkmark}, c'_{\times}, \min(2n, \Delta))$	else if $n < \Delta $ ("increase granularity")
$(c'_{\checkmark}, c'_{\times})$	otherwise

16

16

```
def dd(c_pass, c_fail):
    n = 2
    while 1:
        delta = listminus(c_fail, c_pass)
        deltas = split(delta, n); offset = 0; j = 0
        while j < n:
            i = (j + offset) % n
            next_c_pass = listunion(c_pass, deltas[i])
            next_c_fail = listminus(c_fail, deltas[i])
            if test(next_c_fail) == FAIL and n == 2:
                c_fail = next_c_fail; n = 2; offset = 0; break
            elif test(next_c_fail) == PASS:
                c_pass = next_c_fail; n = 2; offset = 0; break
            elif test(next_c_pass) == FAIL:
                c_fail = next_c_pass; n = 2; offset = 0; break
            elif test(next_c_fail) == FAIL:
                c_fail = next_c_fail; n = max(n - 1, 2); offset = i; break
            elif test(next_c_pass) == PASS:
                c_pass = next_c_pass; n = max(n - 1, 2); offset = i; break
            else:
                j = j + 1
        if j >= n:
            if n >= len(delta):
                return (delta, c_pass, c_fail)
            else:
                n = min(len(delta), n * 2)
```

17

17

Properties

number of tests t – worst case:

$$t = |\Delta|^2 + 7|\Delta| \quad \text{where} \quad \Delta = c_{\times} \setminus c_{\checkmark}$$

number of tests t – best case
(no unresolved outcomes):

$$t \leq \log_2(\Delta)$$

size of difference – no unresolved outcomes

$$|c'_{\times} \setminus c'_{\checkmark}| = 1$$

18

18

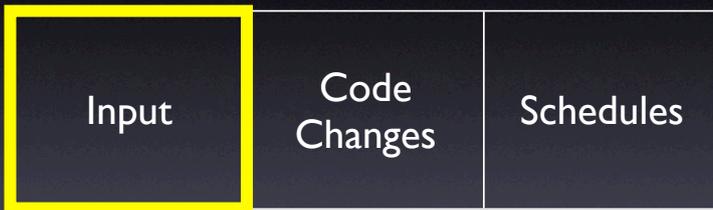
Applications



19

19

Applications



20

20

Isolating Input

```
<SELECT NAME="priority" MULTIPLE SIZE=7>
```



Failure

Isolation: 5 tests
Simplification: 48 tests



21

21

DDInput

The screenshot shows an IDE with a Java test class. A diff window titled "Comparing Passing and Failing Input" is open, showing the following differences:

Passing Input	Failing Input
<SPEAKER>ROMEO</SPEAKER>	<SPEAKER>RO
<LINE>Father, what news? what is the prince	<LINE>Fathe
<LINE>What sorrow craves acquaintance at my	<LINE>What
<LINE>That I yet know not?</LINE>	<LINE>That
</SPEECH>	</SPEECH>
<SPEECH>	<SPEECH>
<SPEAKER>FRIAR LAURENCE</SPEAKER>	<SPEAKER>FR
<LINE>Too familiar</LINE>	<LINE>Too f
<LINE>Is my dear son with such sour company	<LINE>Is my
<LINE>I bring thee tidings of the prince's	<LINE>I br
</SPEECH>	</SPEECH>
<SPEECH>	<SPEECH>
<SPEAKER>ROMEO</SPEAKER>	<SPEAKER>RC
<LINE>What less than	<LINE>What

The IDE interface also shows a "Failure Trace" at the bottom with the following stack trace:

```

junit.framework.AssertionFailedError
at junit.framework.Assert.fail(Assert.java:47)
at junit.framework.Assert.assertTrue(Assert.java:20)
at junit.framework.Assert.assertTrue(Assert.java:27)
at SampleTest.testFileRead(SampleTest.java:46)
  
```

22

Applications

Input

Code
Changes

Schedules

23

23

Applications

Input

Code
Changes

Schedules

24

24

Code Changes

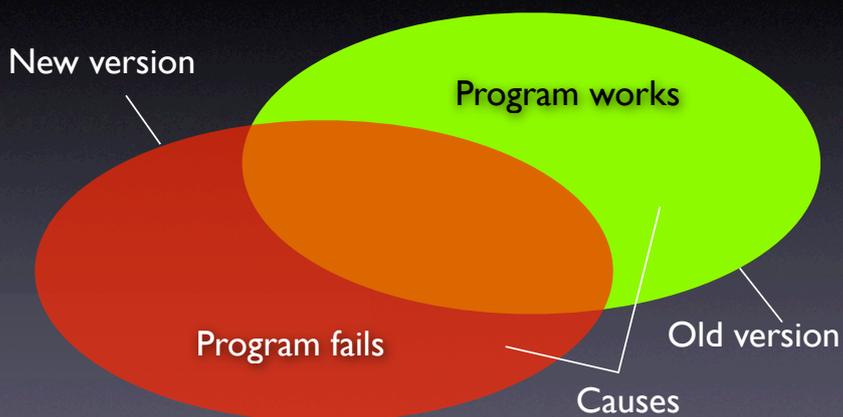
From: Brian Kahne <bkahne@ibm.com>
To: DDD Bug Report Address <bug-ddd@gnu.org>
Subject: Problem with DDD and GDB 4.17

When using DDD with GDB 4.16, the run command correctly uses any prior command-line arguments, or the value of "set args". However, when I switched to GDB 4.17, this no longer worked: If I entered a run command in the console window, the prior command-line options would be lost. [...]

25

25

Version Differences



26

26

What was Changed

```
$ diff -r gdb-4.16 gdb-4.17
diff -r gdb-4.16/COPYING gdb-4.17/COPYING
5c5
< 675 Mass Ave, Cambridge, MA 02139, USA
---
> 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
282c282
< Appendix: How to Apply These Terms to Your New Programs
---
> How to Apply These Terms to Your New Programs
```

...and so on for 178,200 lines (8,721 locations)

27

27

Challenges

- Granularity – within some large change, only a few lines may be relevant
- Interference – some (later) changes rely on other (earlier) changes
- Inconsistency – some changes may have to be combined to produce testable code

Delta debugging handles all this

28

28

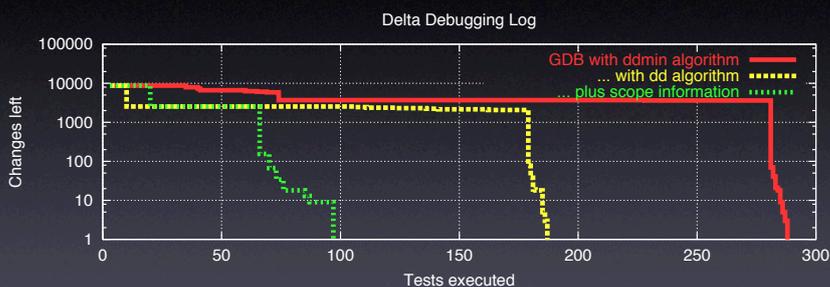
General Plan

- Decompose diff into changes per location (= 8,721 individual changes)
- Apply subset of changes, using PATCH
- Reconstruct GDB; build errors mean unresolved test outcome
- Test GDB and return outcome

29

29

Isolating Changes



- Result after 98 tests (= 1 hour)

30

30

The Failure Cause

```
diff -r gdb-4.16/gdb/infcmd.c gdb-4.17/gdb/infcmd.c
1239c1278
< "Set arguments to give program being debugged when it is
started.\n
---
> "Set argument list to give program being debugged when
it is started.\n
```

- Documentation becomes GDB output
- DDD expects Arguments, but GDB outputs Argument list

31

31

DDChange

The screenshot shows an IDE interface with several panels. The top panel displays a diff of code changes between two versions of a file. The middle panel shows a 'Failure Trace' with a stack trace starting from 'JUnit.Framework.ComparisonFailure'. The bottom panel shows a 'Package Explorer' with a tree view of the project structure. The text 'DDChange' is overlaid on the top part of the screenshot.

32

32

Optimizations

- History – group changes by creation time
- Reconstruction – cache several builds
- Grouping – according to scope
- Failure Resolution – scan error messages for possibly missing changes

33

33

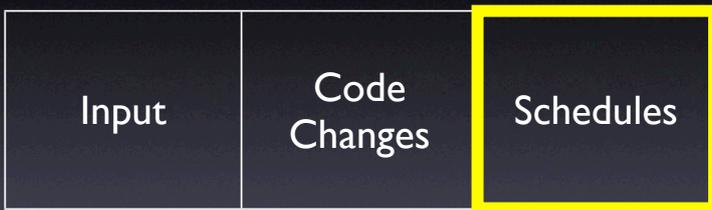
Applications



34

34

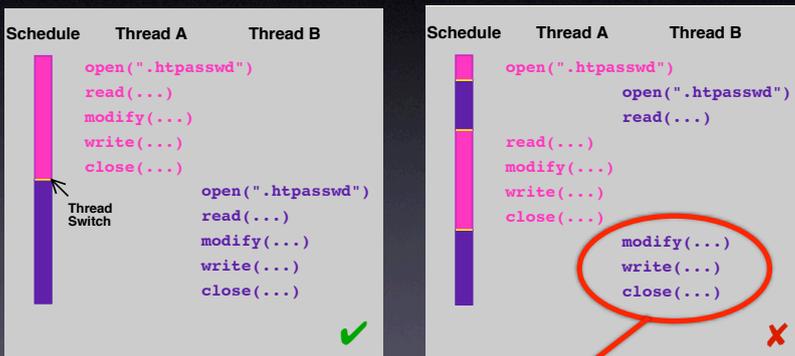
Applications



35

35

Thread Schedules

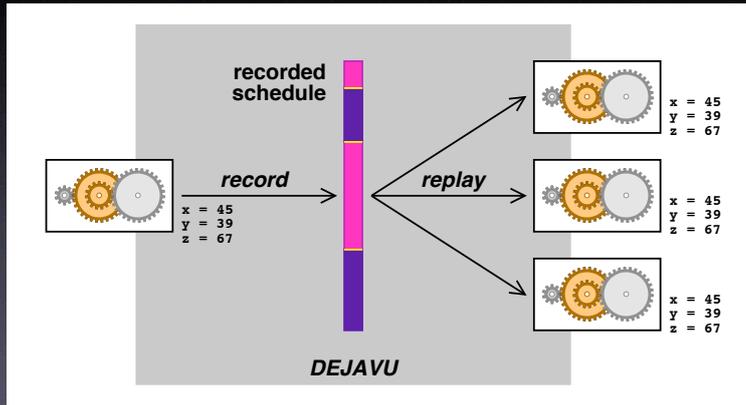


A's updates get lost!

36

36

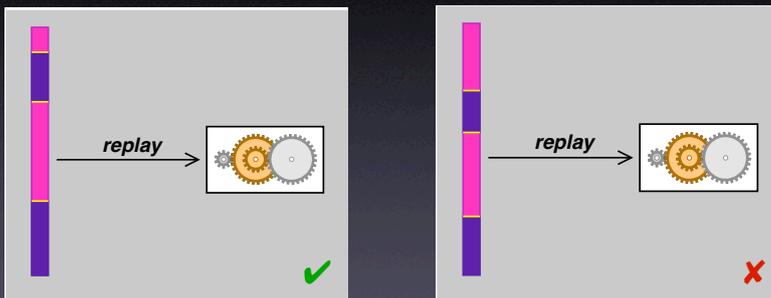
Record + Replay



37

37

Schedules as Input

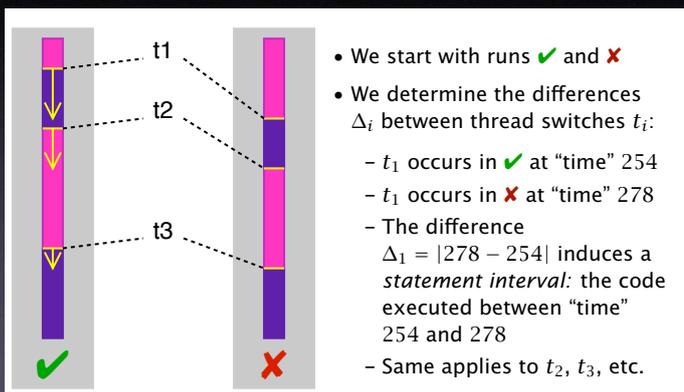


The schedule difference causes the failure!

38

38

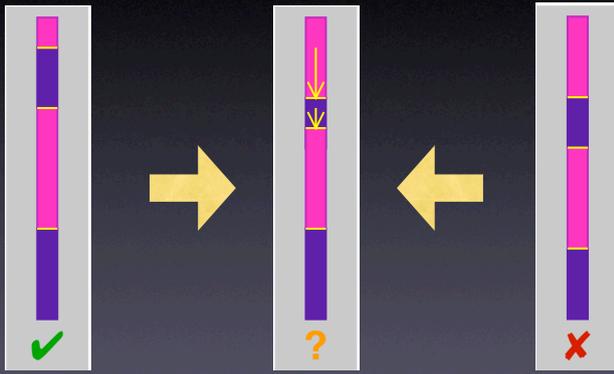
Finding Differences



39

39

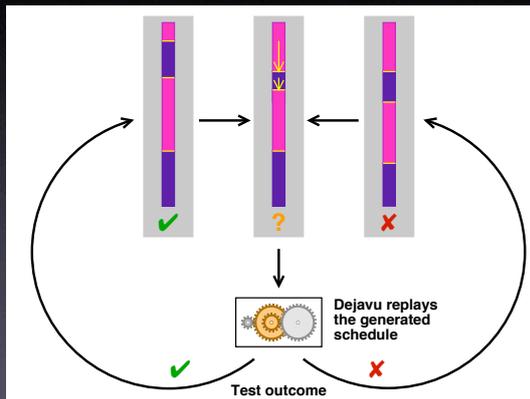
Isolating Differences



40

40

Isolating Differences



41

41

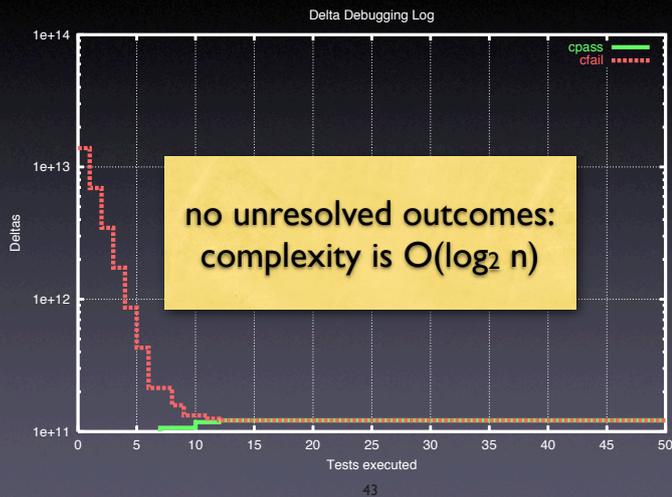
Example: Raytracer

- Raytracer program from Spec JVM98 suite
- Injected a simple *race condition*
- Set up *automated test + random schedules*
- Obtained *passing and failing* schedule
- 3,842,577,240 differences, each moving a thread switch by ± 1 *yield point* (time unit)

42

42

Isolating Schedules



The Failure Cause

```
25 public class Scene { ...
44     private static int ScenesLoaded = 0;
45     (more methods...)
81     private
82     int LoadScene(String filename) {
84         int OldScenesLoaded = ScenesLoaded;
85         (more initializations...)
91         infile = new DataInputStream(...);
92         (more code...)
130        ScenesLoaded = OldScenesLoaded + 1;
131        System.out.println("" +
            ScenesLoaded + " scenes loaded.");
132        ...
134    }
135    ...
733 }
```

44

General Issues

- How do we choose the *alternate world*?
 - How do we *decompose* the configuration?
 - How do we know *a* failure is *the* failure?
 - How do we disambiguate *multiple causes*?
 - How do I get to the defect?
- 45

Concepts

- ★ To isolate failure causes automatically, use
 - an *automated test case*
 - a means to *narrow down the difference*
 - a *strategy* for proceeding.
- ★ One possible strategy is Delta Debugging.

46

46

Concepts (2)

- ★ Delta Debugging can isolate failure causes
 - in the (general) *input*
 - in the *version history*
 - in *thread schedules*
- ★ Every such cause implies a *fix* – but not necessarily a correction.

47

47

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>

or send a letter to Creative Commons, 559 Abbott Way, Stanford, California 94305, USA.

48

48