Simplifying Problems

Andreas Zeller

Simplifying

- Once one has reproduced a problem, one must find out *what's relevant*:
  - Does the problem really depend on 10,000 lines of input?
  - Does the failure really require this exact schedule?
  - Do we need this sequence of calls?

Why simplify?

Simplifying

- For every circumstance of the problem, check whether it is relevant for the problem to occur.
- If it is not, remove it from the problem report or the test case in question.

Circumstances

- Any aspect that may influence a problem is a *circumstance:*
    - Aspects of the problem environment
    - Individual steps of the problem history

Experimentation

- By *experimentation*, one finds out whether a circumstance is relevant or not:
- Omit the circumstance and try to reproduce the problem.
- The circumstance is relevant iff the problem no longer occurs.

Mozilla Bug #24735

Ok the following operations cause mozilla to crash consistently on my machine

-> Start mozilla
-> Go to bugzilla.mozilla.org
-> Select search for bug
-> Print to file setting the bottom and right margins to .50
(I use the file /var/tmp/netscape.ps)
-> Once it's done printing do the exact same thing again on
the same file (/var/tmp/netscape.ps)
-> This causes the browser to crash with a segfault

bugzilla.mozilla.org

```
<SELECT NAME="op_sys" MULTIPLE SIZE=7>
<OPTION VALUE="All">All<OPTION VALUE="Windows 3.1">Windows 3.1<OPTION
VALUE="Windows 95">Windows 95<OPTION VALUE="Windows 98">Windows
98<OPTION VALUE="Windows ME">Windows ME<OPTION VALUE="Windows
2000">Windows 2000<OPTION VALUE="Windows NT">Windows NT<OPTION
VALUE="Mac System 7">Mac System 7<OPTION VALUE="Mac System 7.5">Mac
System 7.5<OPTION VALUE="Mac System 7.6.1">Mac System 7.6.1<OPTION
VALUE="Mac System 8.0">Mac System 8.0<OPTION VALUE="Mac System
8.5">Mac System 8.5<OPTION VALUE="Mac System 8.6">Mac System
8.6<OPTION VALUE="Mac System 9.x">Mac System 9.x<OPTION VALUE="MacOS
X">MacOS X<OPTION VALUE="Linux">Linux<OPTION VALUE="BSDI">BSDI<OPTION
VALUE="FreeBSD">FreeBSD<OPTION VALUE="NetBSD">NetBSD<OPTION
VALUE="OpenBSD">OpenBSD<OPTION VALUE="AIX">AIX<OPTION
VALUE="BeOS">BeOS<OPTION VALUE="HP-UX">HP-UX<OPTION
VALUE="IRIX">IRIX<OPTION VALUE="Neutrino">Neutrino<OPTION
VALUE="OpenVMS">OpenVMS<OPTION VALUE="OS/2">OS/2<OPTION VALUE="OSF/
1">OSF/1<OPTION VALUE="Solaris">Solaris<OPTION
VALUE="SunOS">SunOS<OPTION VALUE="other">other</SELECT>

</td>
<td align=left valign=top>
<SELECT NAME="priority" MULTIPLE SIZE=7>
<OPTION VALUE="--">--<OPTION VALUE="P1">P1<OPTION VALUE="P2">P2<OPTION
VALUE="P3">P3<OPTION VALUE="P4">P4<OPTION VALUE="P5">P5</SELECT>
```

What’s relevant in here?

Why simplify?

- Ease of communication. A simplified test case is easier to communicate.
- Easier debugging. Smaller test cases result in smaller states and shorter executions.
- Identify duplicates. Simplified test cases *subsume* several duplicates.

The Gecko BugAThon

- Download the Web page to your machine.
- Using a text editor, start removing HTML from the page. Every few minutes, make sure it still reproduces the bug.
- Code not required to reproduce the bug can be safely removed.
- When you've cut away as much as you can, you're done.

Rewards

5 bugs - invitation to the Gecko launch party

10 bugs - the invitation, plus an attractive Gecko stuffed animal

12 bugs - the invitation, plus an attractive Gecko stuffed animal autographed by Rick Gessner, the Father of Gecko

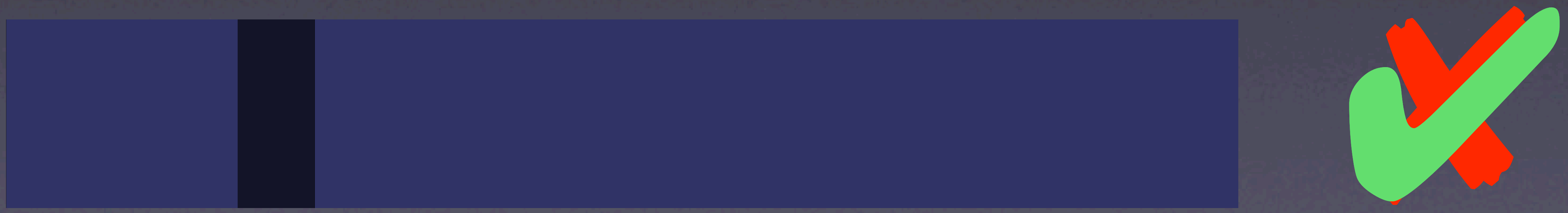
15 bugs - the invitation, plus a Gecko T-shirt

20 bugs - the invitation, plus a Gecko T-shirt signed by the whole raptor team

Binary Search

- Proceed by binary search. Throw away half the input and see if the output is still wrong.
- If not, go back to the previous state and discard the other half of the input.

HTML input



Simplified Input

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

- Simplified from 896 lines to one single line
- Required 12 tests only

Benefits

- Ease of communication. All one needs is "Printing <SELECT> crashes".
- Easier debugging. We can directly focus on the piece of code that prints <SELECT>.
- Identify duplicates. Check other test cases whether they're <SELECT>-related, too.

Why automate?

- Manual simplification is *tedious*.
- Manual simplification is *boring*.
- We have machines for tedious and boring tasks.

Basic Idea

- We set up an *automated test* that checks whether the failure occurs or not
(= Mozilla crashes when printing or not)
- We implement a *strategy* that realizes the binary search.

Automated Test

1. Launch Mozilla
2. Replay (previously recorded) steps from problem report
3. Wait to see whether
   - Mozilla crashes (= the test *fails*)
   - Mozilla still runs (= the test *passes*)
4. If neither happens, the test is *unresolved*

Binary Search

```
<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>   ✓
```

What do we do if *both halves* pass?

Configuration

Circumstance

$$\delta$$

All circumstances

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

Configuration $c \subseteq C$

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

Tests

Testing function

$$test(c) \in \{\text{✔}, \text{✘}, ?\}$$

Failure-inducing configuration

$$test(c_{\text{✘}}) = \text{✘}$$

Relevant configuration $c'_{\text{✘}} \subseteq c_{\text{✘}}$

$$\forall \delta_i \in c'_{\text{✘}} \cdot test(c'_{\text{✘}} \setminus \{\delta_i\}) \neq \text{✘}$$

Binary Strategy

Split input

$$c_{\text{✗}} = c_1 \cup c_2$$

If removing first half fails…

$$test(c_{\text{✗}} \setminus c_1) = \text{✗} \implies c_{\text{✗}}{}' = c_{\text{✗}} \setminus c_1$$

If removing second half fails…

$$test(c_{\text{✗}} \setminus c_2) = \text{✗} \implies c_{\text{✗}}{}' = c_{\text{✗}} \setminus c_2$$

Otherwise, increase granularity:

$$c_{\text{✗}} = c_1 \cup c_2 \cup c_3 \cup c_4$$

$$c_{\text{✗}} = c_1 \cup c_2 \cup c_3 \cup c_4 \cup c_5 \cup c_6 \cup c_7 \cup c_8$$

General Strategy

Split input into *n* parts (initially 2)

$$c_{\text{✘}} = c_1 \cup c_2 \cup \cdots \cup c_n$$

If some removal fails…

$$\exists i \in \{1, \ldots, n\} \cdot \mathit{test}(c_{\text{✘}} \setminus c_i) = \text{✘} \implies \begin{array}{l} {c_{\text{✘}}}' = c_{\text{✘}} \setminus c_i \\ n' = \max(n-1, 2) \end{array}$$

Otherwise, increase granularity

$${c_{\text{✘}}}' = c_{\text{✘}} \quad n' = 2n$$

ddmin in a Nutshell

$c'_{\text{✘}} = \mathit{ddmin}(c_{\text{✘}})$ is a relevant configuration

$\mathit{ddmin}(c_{\text{✘}}) = \mathit{ddmin}'(c'_{\text{✘}}, 2)$ with $\mathit{ddmin}'(c'_{\text{✘}}, n) =$

$$\begin{cases} c'_{\text{✘}} & \text{if } |c'_{\text{✘}}| = 1 \\ \mathit{ddmin}'(c'_{\text{✘}} \setminus c_i, \max(n-1, 2)) & \text{else if } \exists i \in \{1..n\} \cdot \mathit{test}(c'_{\text{✘}} \setminus c_i) = \text{✘} \\ & \text{(“some removal fails”)} \\ \mathit{ddmin}'(c'_{\text{✘}}, \min(2n, |c'_{\text{✘}}|)) & \text{else if } n < |c'_{\text{✘}}| \text{ (“increase granularity”)} \\ c'_{\text{✘}} & \text{otherwise} \end{cases}$$

where $c'_{\text{✘}} = c_1 \cup c_2 \cup \cdots \cup c_n$

$$\forall c_i, c_j \cdot c_i \cap c_j = \emptyset \land |c_i| \approx |c_j|$$

```
def _ddmin(circumstances, n):
    while len(circumstances) >= 2:
        subsets = split(circumstances, n)

        some_complement_is_failing = 0
        for subset in subsets:
            complement = listminus(circumstances, subset)
            if test(complement) == FAIL:
                circumstances = complement
                n = max(n - 1, 2)
                some_complement_is_failing = 1
                break

        if not some_complement_is_failing:
            if n == len(circumstances):
                break
            n = min(n * 2, len(circumstances))

    return circumstances
```

ddmin at Work

Input: **<SELECT NAME="priority" MULTIPLE SIZE=7>** ⟨40 characters⟩ ✘
<SELECT NAME="priority" MULTIPLE SIZE=7> ⟨0 characters⟩ ✔

1 <SELECT NAME="priori**ty" MULTIPLE SIZE=7>** ⟨20⟩ ✔
2 **<SELECT NAME="priori**ty" MULTIPLE SIZE=7> ⟨20⟩ ✔
3 <SELECT NA**ME="priority" MULTIPLE SIZE=7>** ⟨30⟩ ✔
4 **<SELECT NA**ME="priori**ty" MULTIPLE SIZE=7>** ⟨30⟩ ✘
5 <SELECT NAME="priori**ty" MULTIPLE SIZE=7>** ⟨20⟩ ✔
6 **<SELECT NA**ME="priority" MULTIP**LE SIZE=7>** ⟨20⟩ ✘
7 <SELECT NAME="priority" MULTIP**LE SIZE=7>** ⟨10⟩ ✔
8 **<SELECT NA**ME="priority" MULTIPLE SIZE=7> ⟨10⟩ ✔
9 <SELE**CT NA**ME="priority" MULTIP**LE SIZE=7>** ⟨15⟩ ✔
10 **<SELE**CT NAME="priority" MULTIP**LE SIZE=7>** ⟨15⟩ ✔
11 **<SELECT NA**ME="priority" MULTIPLE SI**ZE=7>** ⟨15⟩ ✘
12 <SELE**CT NA**ME="priority" MULTIPLE SI**ZE=7>** ⟨10⟩ ✔
13 **<SELE**CT NAME="priority" MULTIPLE SI**ZE=7>** ⟨10⟩ ✔
14 **<SELECT NA**ME="priority" MULTIPLE SIZE=7> ⟨10⟩ ✔
15 <SE**LECT NA**ME="priority" MULTIPLE SI**ZE=7>** ⟨12⟩ ✔
16 **<SE**LE**CT NA**ME="priority" MULTIPLE SI**ZE=7>** ⟨13⟩ ✔
17 **<SELE**CT **NA**ME="priority" MULTIPLE SI**ZE=7>** ⟨12⟩ ✔
18 **<SELECT** NAME="priority" MULTIPLE SI**ZE=7>** ⟨13⟩ ✘
19 <SE**LECT** NAME="priority" MULTIPLE SI**ZE=7>** ⟨10⟩ ✔
20 **<SE**LE**CT** NAME="priority" MULTIPLE SI**ZE=7>** ⟨10⟩ ✔
21 **<SELEC**T NAME="priority" MULTIPLE SI**ZE=7>** ⟨11⟩ ✔
22 **<SELECT** NAME="priority" MULTIPLE SIZE=**7>** ⟨10⟩ ✘
23 <SE**LECT** NAME="priority" MULTIPLE SIZE=**7>** ⟨7⟩ ✔
24 **<SE**LE**CT** NAME="priority" MULTIPLE SIZE=**7>** ⟨8⟩ ✔
25 **<SELE**CT NAME="priority" MULTIPLE SIZE=**7>** ⟨7⟩ ✔
26 **<SELECT** NAME="priority" MULTIPLE SIZE=7> ⟨8⟩ ✔
27 <**SELECT** NAME="priority" MULTIPLE SIZE=**7>** ⟨9⟩ ✔
28 **<**S**ELECT** NAME="priority" MULTIPLE SIZE=**7>** ⟨9⟩ ✔
29 **<S**E**LECT** NAME="priority" MULTIPLE SIZE=**7>** ⟨9⟩ ✔
30 **<SE**L**ECT** NAME="priority" MULTIPLE SIZE=**7>** ⟨9⟩ ✔
31 **<SEL**EC**T** NAME="priority" MULTIPLE SIZE=**7>** ⟨8⟩ ✔
32 **<SELEC**T NAME="priority" MULTIPLE SIZE=**7>** ⟨9⟩ ✔
33 **<SELECT** NAME="priority" MULTIPLE SIZE=7**>** ⟨8⟩ ✘
34 <**SELECT** NAME="priority" MULTIPLE SIZE=7**>** ⟨7⟩ ✔
35 **<**S**ELECT** NAME="priority" MULTIPLE SIZE=7**>** ⟨7⟩ ✔
36 **<S**E**LECT** NAME="priority" MULTIPLE SIZE=7**>** ⟨7⟩ ✔
37 **<SE**L**ECT** NAME="priority" MULTIPLE SIZE=7**>** ⟨7⟩ ✔
38 **<SEL**E**CT** NAME="priority" MULTIPLE SIZE=7**>** ⟨7⟩ ✔
39 **<SELE**CT NAME="priority" MULTIPLE SIZE=7**>** ⟨6⟩ ✔
40 **<SELECT** NAME="priority" MULTIPLE SIZE=7> ⟨7⟩ ✔
41 <**SELECT** NAME="priority" MULTIPLE SIZE=7**>** ⟨7⟩ ✔
42 **<**S**ELECT** NAME="priority" MULTIPLE SIZE=7**>** ⟨7⟩ ✔
43 **<S**E**LECT** NAME="priority" MULTIPLE SIZE=7**>** ⟨7⟩ ✔
44 **<SE**L**ECT** NAME="priority" MULTIPLE SIZE=7**>** ⟨7⟩ ✔
45 **<SEL**E**CT** NAME="priority" MULTIPLE SIZE=7**>** ⟨7⟩ ✔
46 **<SELE**C**T** NAME="priority" MULTIPLE SIZE=7**>** ⟨7⟩ ✔
47 **<SELEC**T NAME="priority" MULTIPLE SIZE=7**>** ⟨7⟩ ✔
48 **<SELECT** NAME="priority" MULTIPLE SIZE=7> ⟨7⟩ ✔

Result: **<SELECT>**

Complexity

- The maximal number of *ddmin* tests is

$$\frac{(|c_{\text{✘}}|^2 + 7|c_{\text{✘}}|)}{2}$$

Worst Case Details

First phase: every test is unresolved

$$
\begin{aligned}
t &= 2 + 4 + 8 + \cdots + 2|c_{\text{✘}}| \\
&= 2|c_{\text{✘}}| + |c_{\text{✘}}| + \frac{|c_{\text{✘}}|}{2} + \frac{|c_{\text{✘}}|}{4} + \cdots = 4|c_{\text{✘}}|
\end{aligned}
$$

Second phase: testing *last* set always fails

$$
\begin{aligned}
t' &= (|c_{\text{✘}}| - 1) + (|c_{\text{✘}}| - 2) + \cdots + 1 \\
&= 1 + 2 + 3 + \cdots + (|c_{\text{✘}}| - 1) \\
&= \frac{|c_{\text{✘}}|(|c_{\text{✘}}| - 1)}{2} = \frac{|c_{\text{✘}}|^2 - |c_{\text{✘}}|}{2}
\end{aligned}
$$

Binary Search

If

- there is only one failure-inducing circumstance, and
- all configurations that include this circumstance fail,

the number of tests is $t \leq \log_2(|c_{\text{✘}}|)$

More Simplification

Simplified failure-inducing *fuzz* input:

- FLEX crashes on 2,121 or more non-newline characters
- NROFF crashes on “\D^J%0F” or “\302\n”
- CRTPLOT crashes on “t”

Minimal Interaction

Ok the following operations cause mozilla to crash consistently on my machine

-> Start mozilla
-> Go to bugzilla.mozilla.org
-> Select search for bug
-> Print to file setting the bottom and right margins to .50
(I use the file /var/tmp/netscape.ps)
-> Once it's done printing do the exact same thing again on the same file (/var/tmp/netscape.ps)
-> This causes the browser to crash with a segfault

Minimal Interaction

Basic idea:
Apply *ddmin* to recorded user interaction

- To reproduce the Mozilla printing crash:
  - Press *P* while holding *Alt*
  - Press *mouse button 1*
  - Release *mouse button 1*

Optimization

- Caching
- Stop Early
- Syntactic Simplification
- Isolate Differences, not Circumstances

Caching

- Basic idea: store the results of earlier test()
- Saves 8 out of 48 tests in <SELECT> example

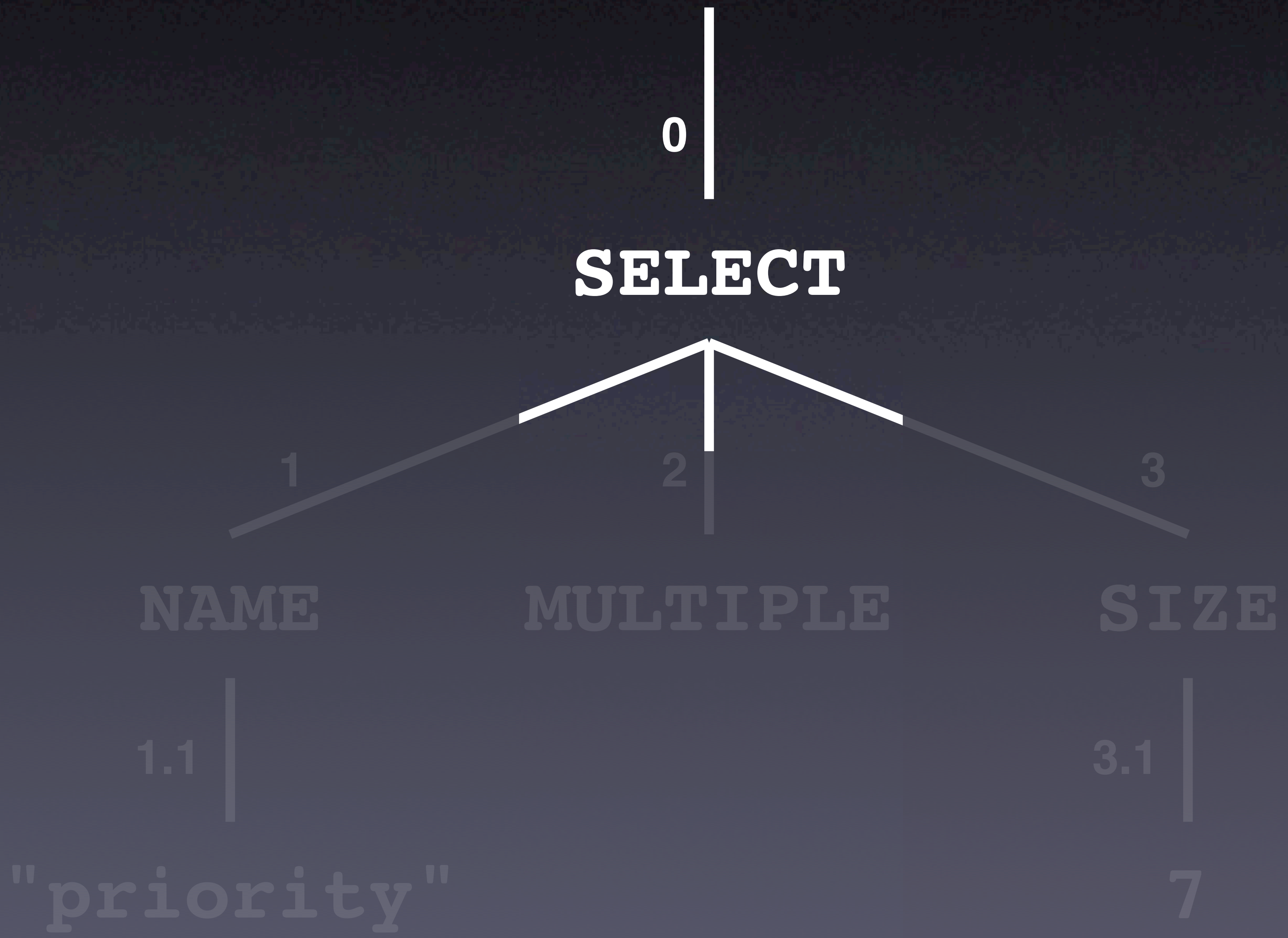
Stop Early

One may stop simplification when

- a certain *granularity* has been reached
- no *progress* has been made
- a certain *amount of time* has elapsed

Syntactic Simplification

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



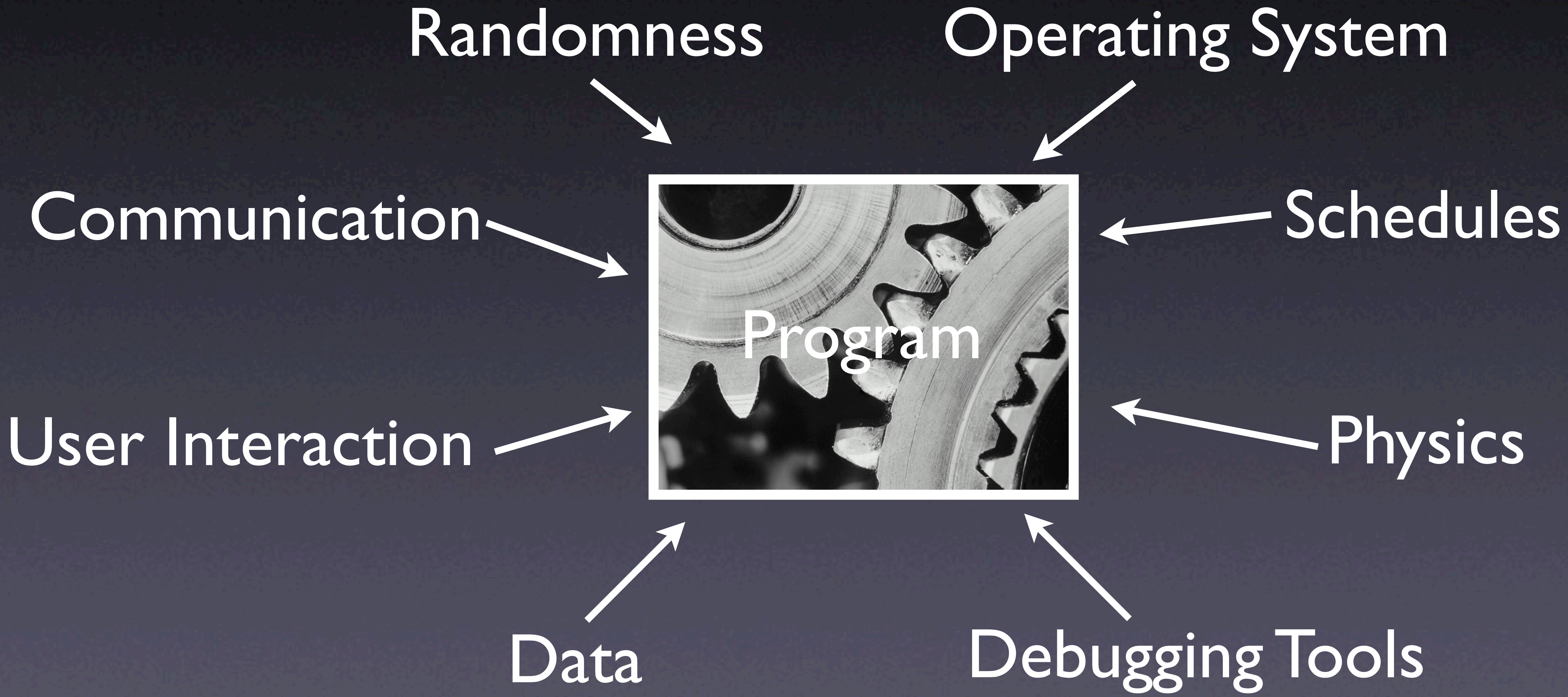
Differences

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

The extra “<” is failure-inducing!

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

More Circumstances



Randomness

Operating System

Communication

Schedules

Program

User Interaction

Physics

Data

Debugging Tools

More Automation

- Failure-Inducing Input
- Failure-Inducing Code Changes
- Failure-Inducing Schedules
- Failure-Inducing Program States

Concepts

- The aim of simplification is to create a simple *test case* from a problem report.
- Simplified test cases…
  - are easier to communicate
  - facilitate debugging
  - identify duplicate problem reports

Concepts (2)

- To simplify a test case, remove all irrelevant circumstances.
- A circumstance is irrelevant if the problem occurs regardless of whether the circumstance is present or not.

Concepts (3)

- To automate simplification, set up
  - an *automated test*
  - a *strategy* to determine the relevant circumstances
- One such strategy is the *ddmin* delta debugging algorithm