

Reproducing Problems

Andreas Zeller



1

The First Task

- Once a problem is reported (or exposed by a test), some programmer must fix it.
- The first task is to *reproduce* the problem.

2

2

Why reproduce?

- **Observing the problem.** Without being able to reproduce the problem, one cannot observe it or find any new facts.
- **Check for success.** How do you know that the problem is actually fixed?

3

3

A Tough Problem

- Reproducing is one of the *toughest* problems in debugging.
- One must
 - recreate the *environment* in which the problem occurred
 - recreate the *problem history* – the steps that lead to the problem

4

4

Reproducing the Environment

Where to reproduce?	Chances of Success	Costs
User	+	--
Developer	○	+

5

5

Iterative Reproduction

- Start with *your* environment
- While the problem is not reproduced, adapt more and more circumstances from the *user's* environment
- Iteration ends when problem is reproduced (or when environments are “identical”)
- Side effect: Learn about failure-inducing circumstances

6

6

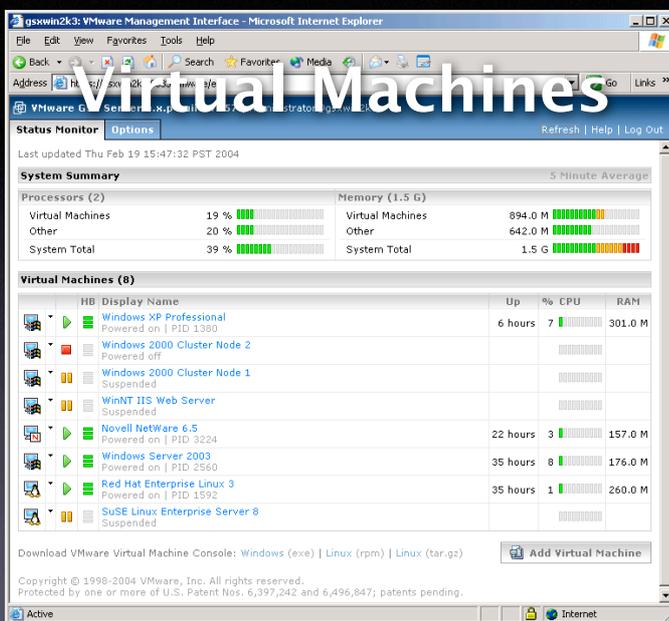
Setting up the Environment

- Millions of configurations
- Testing on dozens of different machines
- All needed to find & reproduce problems

Source: <http://www.ci.newton.ma.us/MIS/Network.htm>

7

Virtual Machines



Source: http://www.vmware.com/products/server/gsx_screens.html

8

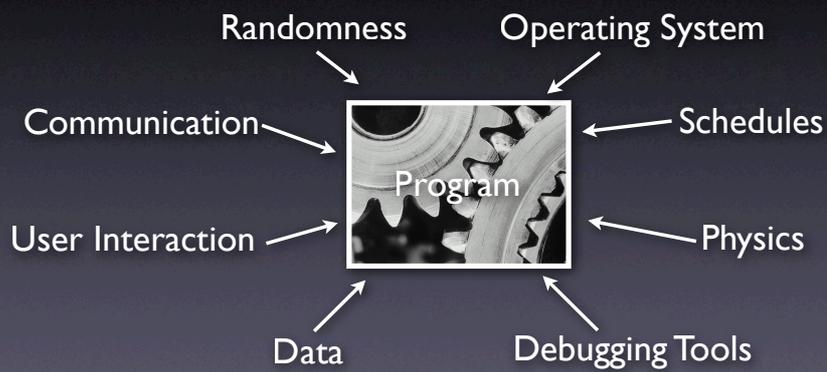
Reproducing Execution

- After reproducing the environment, we must reproduce the *execution*
- Basic idea: Any execution is determined by the *input* (in a general sense)
- Reproducing input → reproducing execution!

9

9

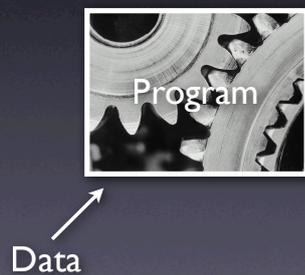
Program Inputs



10

10

Program Inputs



11

11

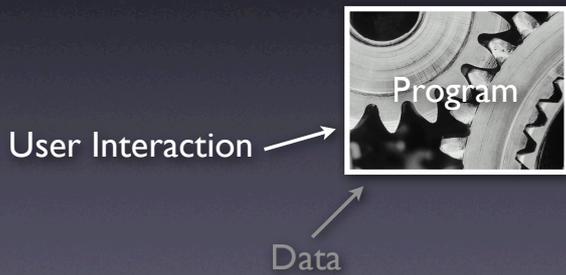
Data

- Easy to transfer and replicate
- Caveat #1: *Get all the data you need*
- Caveat #2: *Get only the data you need*
- Caveat #3: Privacy issues

12

12

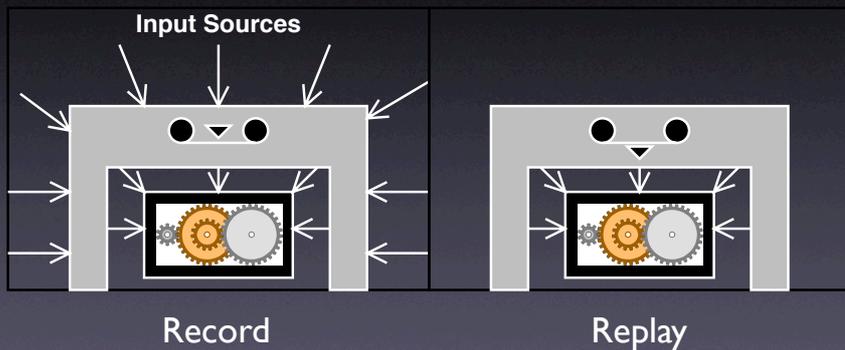
Program Inputs



13

13

User Interaction



14

14

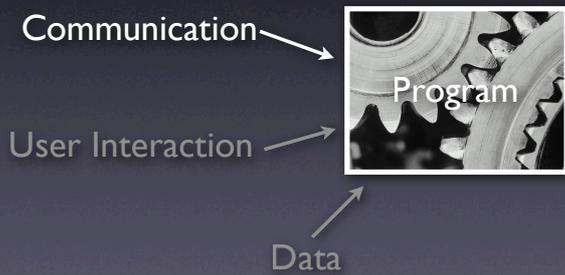
Recorded Interaction

```
send_xevents key H @400,100
send_xevents wait 376
send_xevents key T @400,100
send_xevents wait 178
send_xevents key T @400,100
send_xevents wait 214
send_xevents key P @400,101
send_xevents wait 537
send_xevents keydn Shift_L @400,101
send_xevents wait 218
send_xevents key ";" @400,101
send_xevents wait 167
send_xevents keyup Shift_L @400,101
send_xevents wait 1556
send_xevents click 1 @428,287
send_xevents wait 3765
```

15

15

Program Inputs



16

16

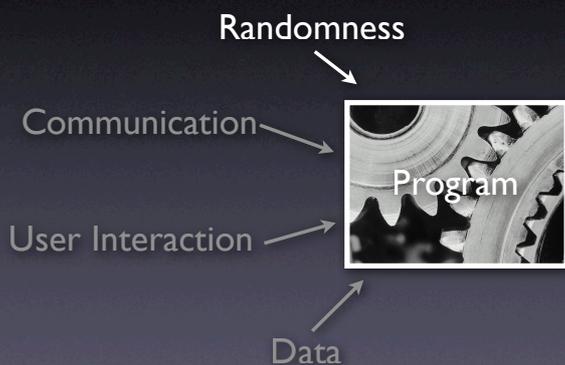
Communication

- General idea: Record and replay like user interaction
- Bad impact on performance
- Alternative #1: Only record since last *checkpoint* (= reproducible state)
- Alternative #2: Only record "last" transaction

17

17

Program Inputs



18

18

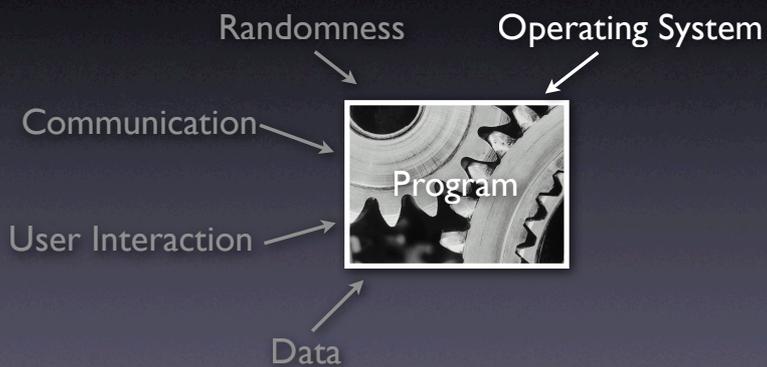
Randomness

- Program behaves different in every run
- Based on random number generator
 - Pseudo-random: save *seed* (and make it configurable)
 - Same applies to *time of day*
 - True random: record + replay sequence

19

19

Program Inputs



20

20

Operating System

- The OS handles *entire* interaction between program and environment
- Recording and replaying OS interaction thus makes entire program run reproducible

21

21

A Password Program

```
#include <string>
#include <iostream>
using namespace std;

string secret_password = "secret";

int main()
{
    string given_password;
    cout << "Please enter your password: ";
    cin >> given_password;
    if (given_password == secret_password)
        cout << "Access granted." << endl;
    else
        cout << "Access denied." << endl;
}
```

```
$ c++ -o password password.C
$ ./password
Please enter your
password: secret
Access granted.
$
```

22

22

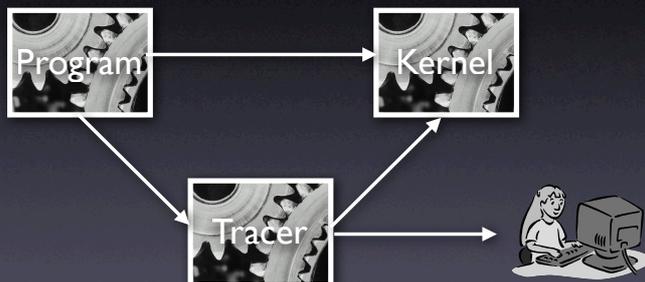
Traced Interaction

```
$ c++ -o password password.C
$ strace ./password 2> LOG
Enter your password: secret
Access granted.
$ cat LOG
...
write(1, "Please enter your password: ", 28) = 28
read(0, "secret\n", 1024) = 7
write(1, "Access granted.\n", 16) = 16
exit_group(0) = ?
```

23

23

How Tracing works



24

24

Replaying Traces



25

25

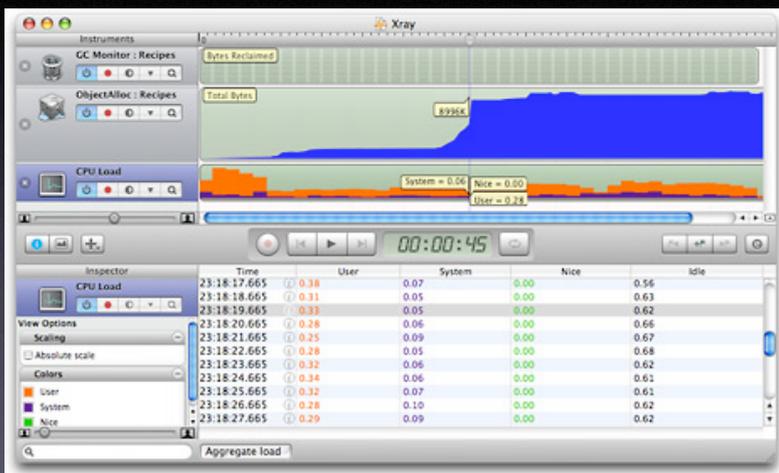
Challenges

- Tracing creates *lots* of data
- Example: Web server with 10 requests/sec
A trace of 10 k/request means 8GB/day
- All of this must be *replayed* to reproduce the failure (alternative: *checkpoints*)
- Huge performance penalty!

26

26

XRay + DTrace



27

27

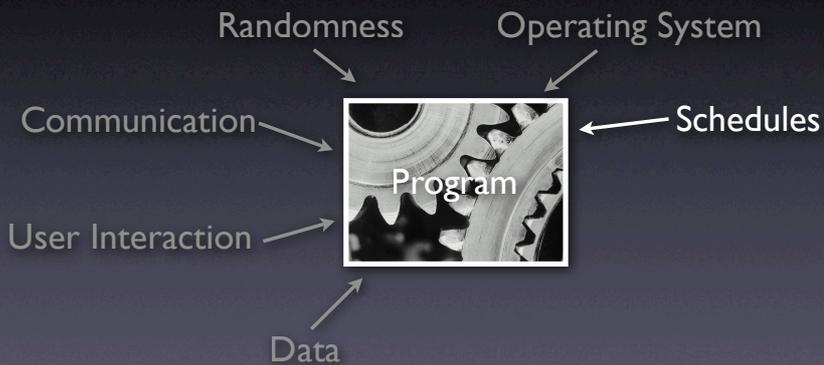
XRay + DTrace

- DTrace: Kernel extension for capturing data
- System interaction can be *monitored*
- Captured I/O can be *replayed* at will
- Focus on high performance

28

28

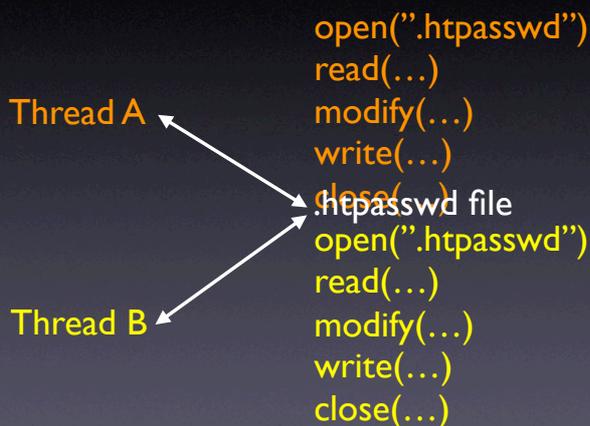
Program Inputs



29

29

Accessing Passwords



30

30

Lost Update

Thread A

```
open(".htpasswd")  
open(".htpasswd")  
read(...)  
read(...)  
modify(...)  
write(...)  
close(...)  
modify(...)  
write(...)  
close(...)
```

A's updates
get lost!

Thread B

31

31

Reproducing Schedules

- Thread changes are induced by a *scheduler*
- It suffices to record the schedule (i.e. the moments in time at which thread switches occur) and to replay it
- Requires deterministic input replay

32

32

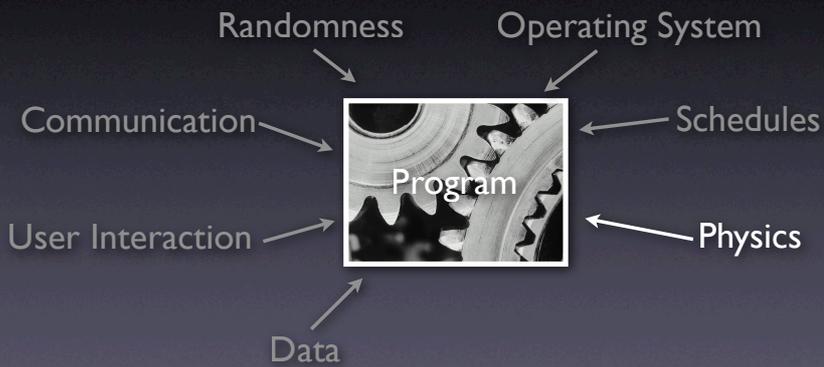
Constructive Solutions

- Lock resource before writing
- Check resource update time before writing
- ... or any other *synchronization mechanism*

33

33

Program Inputs



34

34

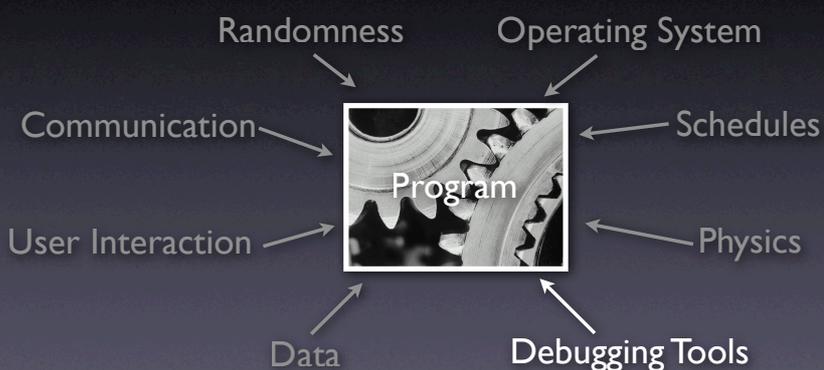
Physical Influences

- Static electricity
 - Alpha particles (*not* cosmic rays)
 - Quantum effects
 - Humidity
 - Mechanical failures + real bugs
- Rare and hard to reproduce

35

35

Program Inputs



36

36

A Heisenbug

- Code fails outside debugger only

```
int f() {  
    int i;  
    return i;  
}
```

In program:
returns random value

In debugger:
returns 0

37

37

More Bugs

Bohr Bug	Heisenbug
Mandelbug	Schrödinbug

38

38

Bohr Bug = Repeatable under well-def'd conditions

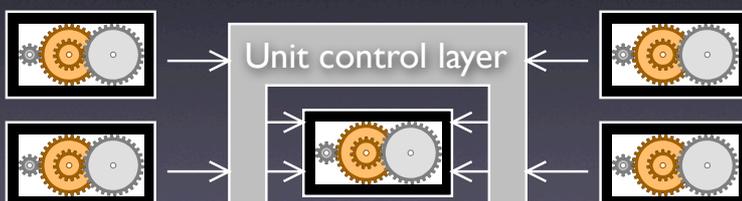
Heisenbug = Changes when observed

Mandelbug = Causes are complex and chaotic, appears non-deterministic, but isn't

Schrödinbug = Never should have worked, and promptly fails as soon one realizes this

Isolating Units

- Capture + replay *unit* instead of program
- Needs an *unit control layer* to monitor input



39

39

Isolated Units

- **Databases.** Replay only the interaction with the database.
- **Compilers.** Record + replay intermediate data structures rather than the entire front-end.
- **Networking.** Record + replay communication calls.

40

40

A Control Example

```
class Map {
public:
    virtual void add(string key, int value);
    virtual void del(string key);
    virtual int lookup(string key);
};
```

- How do we control this?

41

41

A Log as a Program

```
#include "Map.h"
#include <assert>

int main() {
    Map map;
    map.add("onions", 4);
    map.del("truffels");
    assert(map.lookup("onions") == 4);
    return 0;
}
```

- This is a log file (and also a program)
- How do we get this?

42

42

Controlled Map

```
class ControlledMap: public Map {
public:
    typedef Map super;

    virtual void add(string key, int value);
    virtual void del(string key);
    virtual int lookup(string key);

    ControlledMap();           // Constructor
    ~ControlledMap();         // Destructor
};
```

43

43

Logging

```
void ControlledMap::add(string key, int value) {
    clog << "map.add(\"" << key << "\", "
        << value << ");" << endl;
    Map::add(key, value);
}
map.add("onions", 4);

void ControlledMap::del(string key) {
    clog << "map.del(\"" << key << "\");" << endl;
    Map::del(key);
}
map.del("truffels");

virtual int ControlledMap::lookup(string key) {
    clog << "assert(map.lookup(\"" << key << "\") == ";
    int ret = Map::lookup(key);
    clog << ret << ");" << endl;
    return ret;
}
assert(map.lookup("onions") == 4);
```

44

44

Logging Fixture

```
ControlledMap::ControlledMap()
{
    clog << "#include \"Map.h\"" << endl
        << "#include <assert>" << endl
        << "" << endl
        << "int main() {" << endl
        << "    Map map;" << endl;
}

ControlledMap::~ControlledMap()
{
    clog << "    return 0;" << endl;
    << "}" << endl;
}
```

45

45

More Interaction

- Variables (hard to detect)
- Other units (break dependency if needed)
- Time (record + replay, too)

46

46

Mock Objects

- A *Mock Object* simulates an original object
- Its implementation tells how to react on specific calls (i.e. returning other mock objects)
- Can be combined with *recording*, too!

47

47

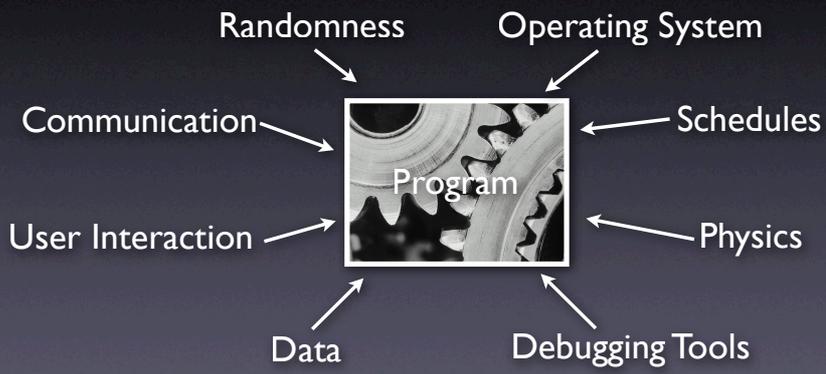
Concepts

- ★ Once a problem is tracked, one must *reproduce it* in the own environment
- ★ To reproduce a problem...
 - reproduce the *environment* (by adopting one circumstance after the other)
 - reproduce the *execution* (by controlling the input of the program or a unit)

48

48

Program Inputs



49

49

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.

50

50