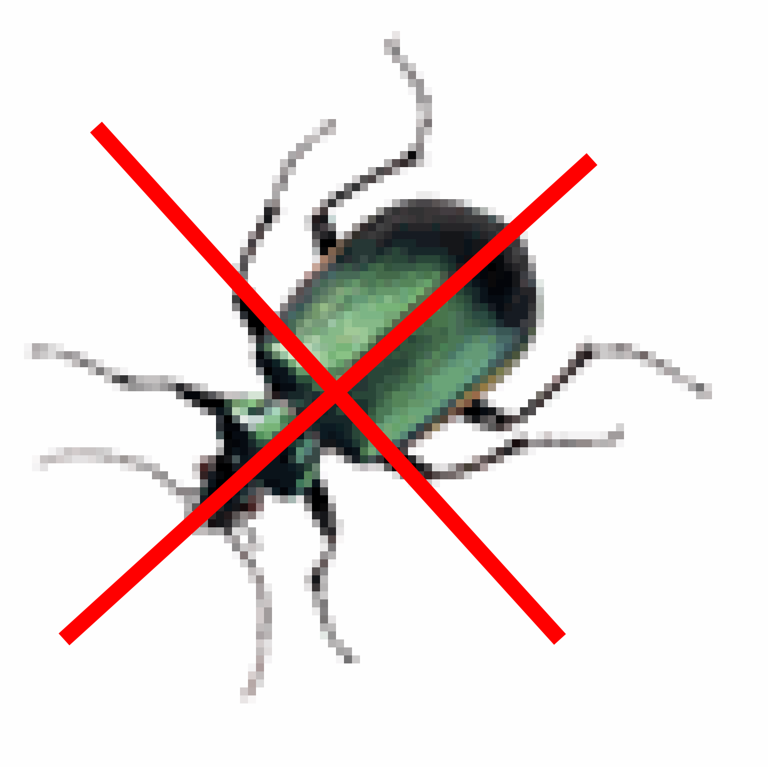


Debugging

Nigel Horspool

(with some material by Mike Zastre)

University of Victoria

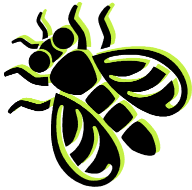
Topics

- Bug Detection Approaches
- Debugging Tools

Bug Detection – Step 1

- Study the failure symptoms to identify (if possible)
  - where in the program the fault occurred and
  - what the program state must have been.

- If necessary add an exception handler to the program (or use a debugging tool), to determine exactly where the error occurs.



Bug Detection – Step 2

- Read and re-read your code in the vicinity of the error location.
  - Better still, have *someone else* read your code.
  - Alternatively find a person who will listen and explain the program’s logic to him or her.

Bug Detection – Step 3

- Experiment to find the simplest / shortest input which causes the failure to occur.
- Disable compiler optimization (if enabled).
- And try to develop theories about the program state which, if reached, would case the observed failure.

These preliminaries will make life much easier for the following steps.

Bug Detection – Step 4

We can try inserting …

- assertions at strategic points to verify that an erroneous program state is not being entered, and/or
- output statements at strategic points to determine whether variables are taking on erroneous values.

Bug Detection

Perhaps the preceding steps have helped track down the error?

But maybe not …

- There is too much trace output to examine?
- We didn’t capture the right information?
- The bug went away when we inserted those extra statements????

The "Heisenbug"

- It is a bug which changes or goes away when you try to observe it.
- How can this happen?
- The addition of code changes the addresses of variables and subroutines, and may affect the positions of objects on the heap and times when a garbage collection occurs. An array indexing error or an invalid pointer may access a different data item and lead to different behaviour …

Heisenberg in 1927

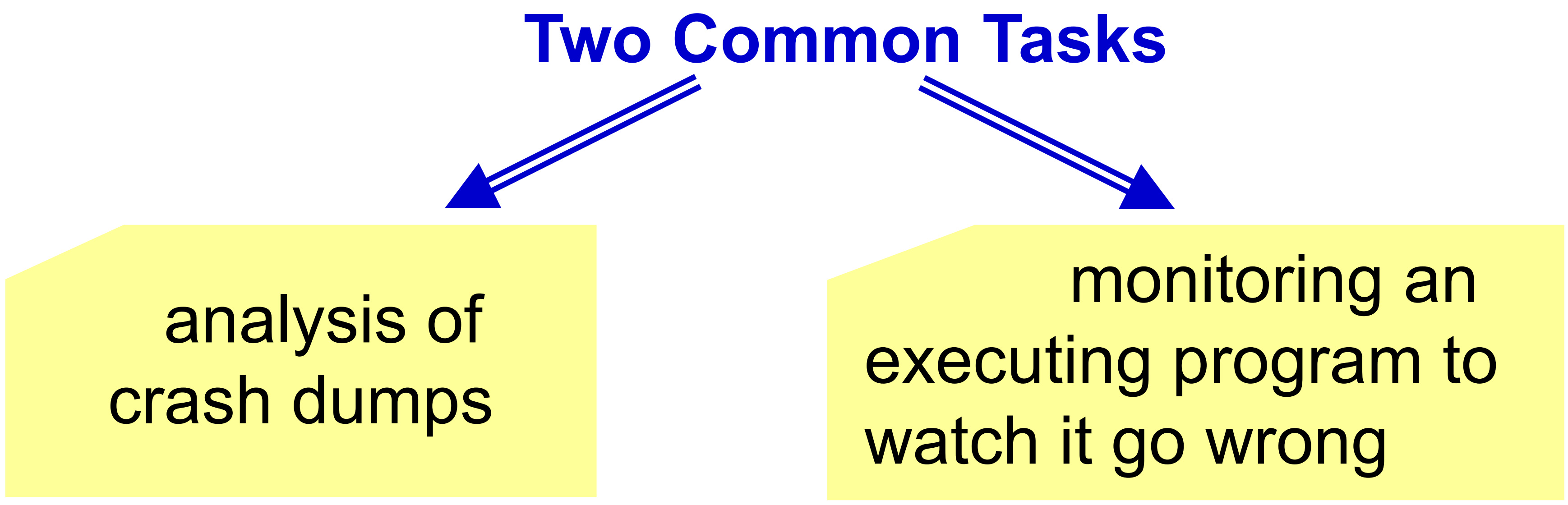
The Debugger

Sometimes we need to observe our bug in an unchanged copy of the program.

We need to have one program watch another program.

The watcher is called a *debugger* (and the program being monitored is the *debuggee*).

The Debugger



Two Common Tasks

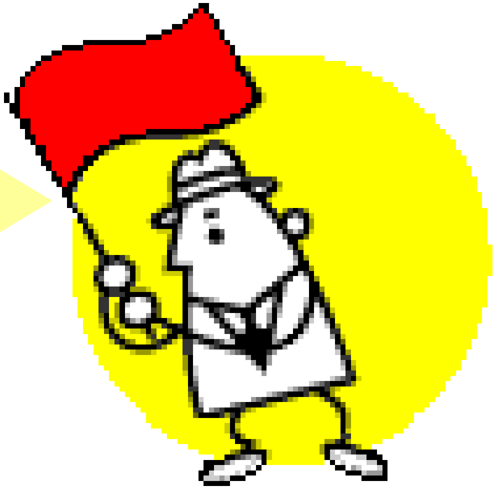
analysis of crash dumps

monitoring an executing program to watch it go wrong

Often, a single tool will perform either task

The Debugger – Note!

There is **no** guarantee that the debuggee will behave in exactly the same way with a debugger



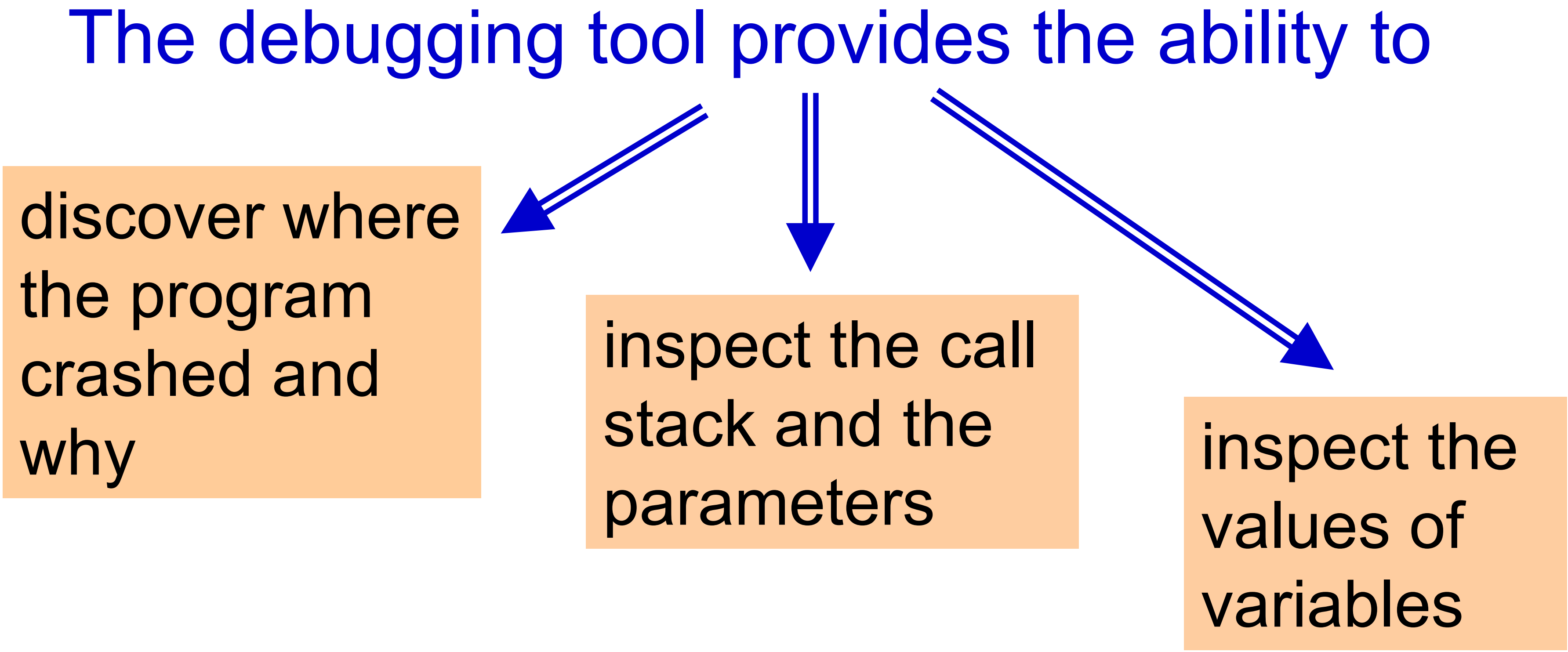
Perhaps it’s a timing dependency? Or perhaps the debuggee’s memory layout is different?

Some Debuggers

- Visual Studio on Windows
- dbx and gdb for debugging C/C++ … on Unix
- eclipse for debugging Java (and other languages)
- jdb for debugging Java
- IDLE for debugging Python

- *insight* is a GUI front-end for for gdb
- *DDD* (Data Display Debugger) is a GUI front-end for gdb/dbx/jdb/…

Analysis of Crash Dumps



The debugging tool provides the ability to

discover where the program crashed and why

inspect the call stack and the parameters

inspect the values of variables

Note: the first two features are provided by a stack trace when an exception is caught in Java or C#.

From Static to Dynamic …

Static Tools	Dynamic Tools
Source code analysis (to check compliance to coding standards, warn about possible problems)	Resource usage monitoring (CPU and memory profilers) e.g. jstat, jconsole, jprofiler, Prof-It, …
Crash dump inspection	**Debuggers**

Common Monitoring Facilities Provided by Debuggers

- Breakpoints
- Single step execution
- Inspect variables, location
- Display source code
- Fault detection
- Watchpoints
- Multithread, multiprocess support

Common Monitoring Facilities Provided by Debuggers

Breakpoints

```
(gdb) break main

(gdb) break 22
```

Single step execution

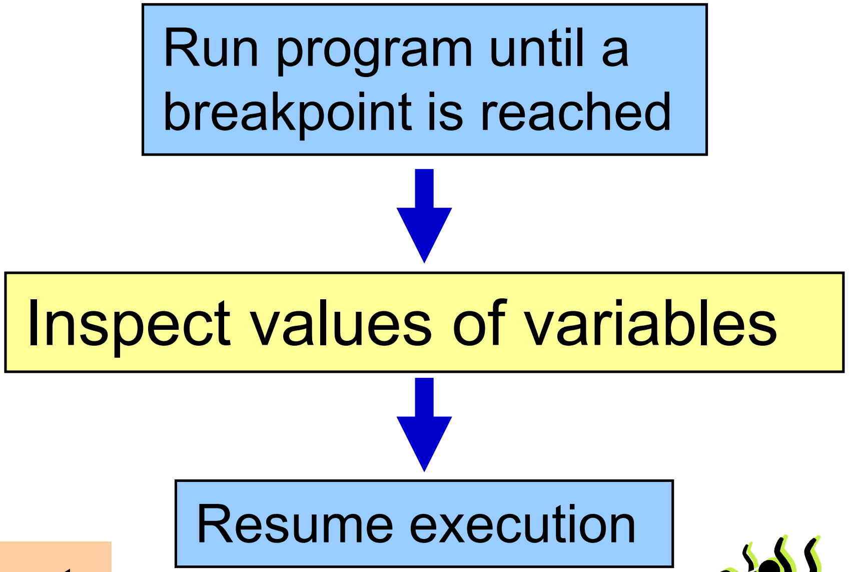
Inspect variables, location

Display source code

Fault detection

Watchpoints

Multithread, multiprocess support


Run program until a breakpoint is reached

Inspect values of variables

Resume execution

Common Monitoring Facilities Provided by Debuggers

- Breakpoints
- Single step execution
- Inspect variables, location
- Display source code
- Fault detection
- Watchpoints
- Multithread, multiprocess support

`step`	execute one statement and stop again
`next`	execute one statement or a complete method/function call and stop again

Common Monitoring Facilities Provided by Debuggers

Breakpoints

Single step execution

Inspect variables, location

```
(gdb) print sum
(gdb) where
(gdb) up
```

Display source code

Fault detection

Watchpoints

Multithread, multiprocess support

Common Monitoring Facilities Provided by Debuggers

- Breakpoints
- Single step execution
- Inspect variables, location
- Display source code
- Fault detection
- Watchpoints
- Multithread, multiprocess support

```
(gdb) list main

(gdb) list encode

(gdb) list 22
```

Common Monitoring Facilities Provided by Debuggers

Breakpoints

Single step execution

Inspect variables, location

Display source code

Fault detection

If the debuggee throws an exception or performs some illegal action, the debugger must re-take control and give the user a chance to inspect the situation

Watchpoints

Multithread, multiprocess support

Common Monitoring Facilities Provided by Debuggers

Breakpoints

Single step execution

Inspect variables, location

Display source code

Fault detection

Watchpoints

```
watch sum
```

watches for accesses to a variable, breaks when value in variable is changed

Multithread, multiprocess support

Common Monitoring Facilities Provided by Debuggers

Breakpoints

Single step execution

Inspect variables, location

Display source code

Fault detection

Watchpoints

Multithread, multiprocess support

Ability to monitor execution of individual threads or processes in the debuggee

Styles of Debuggers

Command-Line

- gdb and jdb are examples

Graphical User Interface

- Visual Studio, eclipse, IDLE, gdb+DDD are examples

Some Further Reading on Debuggers/Debugging

- Adam Barr. ***Find The Bug***. Addison Wesley, 2005. **`http://www.findthebug.com/`**
- E. Allen. ***Bug Patterns in Java***. Apress, 2002.
- S.L. Bartlett, A.R. Ford, T.J. Teorey, G.S. Tyson. ***Practical Debugging in Java***. Pearson, 2004.
- J.B. Rosenberg. ***How Debuggers Work***. Wiley, 1996.
- D.J. Agans. ***Debugging***. AMA, 2002.
- M. Telles, Y. Hsieh. ***The Science of Debugging.*** Coriolis, 2001.