Observing Facts
Andreas Zeller

Reasoning about Runs

- Experimentation: $n$ controlled runs
- Induction: $n$ runs
- Observation: 1 run
- Deduction: 0 runs

Reasoning about Runs

- Observation: 1 run
- Deduction: 0 runs
Principles of Observation

• Don’t interfere.
• Know what and when to observe.
• Proceed systematically.

Logging execution

• General idea: Insert output statements at specific places in the program
• Also known as printf debugging

Printf Problems

• Clobbered code
• Clobbered output
• Slow down
• Possible loss of data (due to buffering)
Better Logging

- Use standard formats
- Make logging optional
- Allow for variable granularity
- Be persistent

Logging Functions

- Have specific functions for logging (e.g. dprintf() to print to a specific logging channel)
- Have specific macros that can be turned on or off—for focusing as well as for production code

Logging Frameworks

- Past: home-grown logging facilities
- Future: standard libraries for logging
- Example: The LOGFORJ framework

Again, demonstrate the use of LOG() interactively
The core idea of LOGFORJ is to assign each class in an application an individual or common logger. A logger is a component which takes a request for logging and logs it. Each logger has a level, from DEBUG over INFO, WARN, and ERROR to FATAL (very important messages).

The core idea of LOGFORJ is to assign each class in an application an individual or common logger. A logger is a component which takes a request for logging and logs it. Each logger has a level, from DEBUG over INFO, WARN, and ERROR to FATAL (very important messages).
Logging with Aspects

- Basic idea: Separate concerns into individual syntactic entities (aspects)
- Aspect code (advice) is woven into the program code at specific places (join points)
- The same aspect code can be woven into multiple places (pointcuts)

A Logging Aspect

```java
public aspect LogBuy {
  pointcut buyMethod():
    call(public void Article.buy());
  before(): buyMethod() {
    System.out.println("Entering Article.buy()")
  }
  after(): buyMethod() {
    System.out.println("Leaving Article.buy()")
  }
}
$ ajc logBuy.aj Article.java
$ java Article
```

Using Pointcuts

```java
public aspect LogArticle {
  pointcut allMethods():
    call(public * Article.*(..));
  before(): allMethods() {
    System.out.println("Entering " + thisJoinPoint)
  }
  after(): allMethods() {
    System.out.println("Leaving " + thisJoinPoint)
  }
}
```
Aspect Arguments

```java
public aspect LogMoves {
    pointcut setP(Line a_line, Point p):
        call(void a_line.setP*(p));

    after(Line a_line, Point p): setP(a_line, p) {
        System.out.println(a_line + " moved to " + p + ".");
    }
}
```

Observation Tools

- Getting started fast – without altering the program code at hand
- Flexible observation of arbitrary events
- Transient sessions – no code is written

Debuggers

- Execute the program and make it stop under specific conditions
- Observe the state of the stopped program
- Change the state of the program
static void shell_sort(int a[], int size)
{
    int i, j;
    int h = 1;
    do {
        h = h * 3 + 1;
    } while (h <= size);
    do {
        h /= 3;
        for (i = h; i < size; i++)
        {
            int v = a[i];
            for (j = i; j >= h && a[j - h] > v; j -= h)
            {
                a[j] = a[j - h];
            }
            if (i != j)
                a[j] = v;
        }
    } while (h != 1);
}

A Debugging Session

More Features

• Control environment
• Post mortem debugging
• Logging data
• Fix and continue

More on Breakpoints

• Data breakpoints (watchpoints)
• Conditional breakpoints

Show this interactively with GDB or DDD

Demonstrate watchpoints and conditionals interactively
Debugger Caveats

• A debugger is a tool, not a toy!

Visualizing Data

Again, demonstrate DDD interactively

Concepts

★ Logging functions ("printf debugging") are easy to use, but clobber code and output

★ To encapsulate and reuse debugging code, use dedicated logging functions or aspects
Concepts (2)

⭐ Logging functions can be turned on or off (and may even remain in the source code)
⭐ Aspects elegantly keep all logging code in one place
⭐ Debuggers allow flexible + quick observation of arbitrary events

Concepts (3)

⭐ To observe the final state of a crashing program, use a debugger
⭐ Advanced debuggers allow to query events in a declarative fashion…
⭐ …as well as visualizing events and data