Debugging Concepts

Philip Johnson
Collaborative Software Development Laboratory
Information and Computer Sciences
University of Hawaii
Honolulu HI 96822

(1)

Issues

What and why of debugging.

Design-level approaches:
- Extreme debugging
- Assertions
- Static Tracing

Dynamic tracing
- Concepts
- Jbuilder example

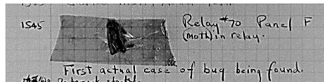
Miscellaneous issues
- Ablation
- Exceptions
- Profiling tools
- Bug reporting
- Domain specific debugging
- First and last resorts

Relative Efficiencies of:
- Extreme Debugging
- Static Tracing
- Dynamic Tracing

(2)

What is a bug?

Reasonable definition:
- A bug is "software behavior that is not intended by the developer".
- In 1945, a large vacuum-tube computer stopped working. After much effort, the problem was identified as a moth getting into the computer by Grace Hopper.
- http://ei.cs.vt.edu/~history/Bug.GIF



(3)

Cost of debugging

Debugging may consume 60-70% of development time

Debugging may be responsible for 80% of all cost overruns.

(4)

Common Bug Types

Compilation/syntax errors:
- Program doesn't compile due to types or environment configuration (bad classpath).

Logic errors:
- Program executes, but output/behavior wrong.

Run-time errors:
- Exceptions thrown (Null pointers, etc.)

Threading errors:
- Multiple threads that do not interact correctly.
- Very difficult to reproduce and track down.

(5)

Motivation for design-level approaches to debugging

It's 3am. The system crashed three hours ago. Your boss says the company is losing $1000 for each minute the system is down. She wants to know what the current status is.

(6)

Answer 1

I've looked and looked and tried everything and I haven't got any ideas left!!

(7)

Answer 2

My analysis shows that classes Foo, Bar, and Baz are working to specification.

Class Qux is suspect, and I'm currently implementing additional tests to verify its non-involvement.

If Qux checks out, the next set of candidates to investigate will be Zob, Doofus, and Chosus.

If you have other developers to spare, they could start on Zob right now.

Do you have any coffee?

(8)

Morals

Design your code to facilitate debugging later.

Develop your software in such a way as to facilitate debugging later.

Learn to use debugging tools; don't try to dig a large hole with a spoon.

(9)

Where does it fail?

The first question to answer in debugging.

In many cases, getting an answer to this question is the primary work required to remove the error.

General procedure:

- Form hypotheses about where failure occurs.
- Create repeatable failure.
- Test your hypotheses.
- Determine if the "test" has long-term value to system.

(10)

Extreme Debugging

1. Design system in modules, each of which has clearly defined behavior.

2. Write JUnit tests to verify that modules exhibit defined behavior.

3. When system fails, debug by:
   - Writing additional Junit tests to improve verification until test failure identifies location.
   - Refactor classes to have more clearly identifiable and testable behavior.

(11)

Extreme Debugging Advantages

System becomes easier to debug over time.

System structure becomes incrementally more clear, understandable, and modifiable.

No expectation that you'll do things perfectly at first.

Exploit bugs as opportunities to improve system design, not just "move on to next bug as quickly as possible"

(12)

Assertions

Assertions are boolean expressions that define the correct state of your program at particular locations in the code.

Format:
- <assertion type>
  - Purpose of assertion
- <condition>
  - Boolean expression testing assertion
- <message>
  - What information to output if failure

(13)

Assertion types

Preconditions:
- Define conditions that must hold when calling a particular method.

Postconditions:
- Define what the method does. Evaluated when method exits.

Invariants:
- Define state-space consistency: things that must always be true. Evaluated at entry and exit points.

Data assertions:
- Define conditions that must hold at the point in the code they are evaluated.

(14)

Goals of assertions

Effectively used assertions can help:
- Locate methods containing bugs
- Speed up debugging
- Provide useful documentation of system.
- Speed up system extension.
- Improve modularity and design.
- Improve understandability.

Assertion technology should be:
- Powerful, easy to write, easy to use
- Not introduce excessive run-time overhead.
- This is not easy to pull off!

Eiffel is best example of assertion technology integrated into a language.

(15)

Issues in assertion language design

Configuration:
- Allowing different assertions to run at different times.

Results:
- Assertion output to file, console, both?
- Exit program? Throw exception?

Placement of postconditions and invariants:
- Methods can exit in multiple places.
- Only one postcondition should suffice for all possible exit points.
- Cannot be accomplished in Java without a preprocessor.

(16)

Assertions in Java 1.4

New keyword added to language: "assert"
- `assert expression1;`
- `assert expression1 : expression2`

Example:
- `assert i >= -1 && i < 256;`

If expression1 is false, then an AssertionError is thrown, with expression2 evaluated and passed to it (if available).

Expression2 not evaluated if expression1 true.

Use –enableassertions on command line to start them; can be enabled for specific packages and classes

(17)

Java 1.4 Assertion Design FAQ

Why language changed vs. library:
- Enable efficient assertions (no run-time overhead without IF statement).

Why not "real" assertion language (I.e. support for postconditions with multiple exits)
- Would force too many changes to Java.

Why not provide compiler flag to eliminate assertions?
- Firm requirement was to enable assertions in the field.

Assertions should not have side-effects
- Since enabled during development; disabled in field during normal operation.

(18)

Tracing

Tracing provides a way to visualize program execution.

Two basic approaches:
- •Static tracing (log files)
- •Dynamic tracing (debuggers)

Approaches are complementary and you should know when to use one and when to use the other.

(19)

Static Tracing Overview

Static tracing (log output):
- •Add explicit trace code to your program.
- •Advantages:
  - -Easily available once implemented
  - -Provides historical information after crash
  - -Can provide exactly right information.
  - -Extremely important for servers and in-the-field debugging

Disadvantages:
- •Adds overhead to code
- •Adds overhead to execution
- •Requires advance thought about what information you will need to know if an error occurs.
- •Not as flexible as dynamic tracing

(20)

Dynamic Tracing Overview

Use of interactive debugger in a development environment.

Advantages:
- •Can step through individual lines of code
- •Can see individual data values
- •Requires no advance thought about what values are important to know about.

Disadvantages:
- •Time consuming.
- •Can easily become an unfocussed "fishing expedition".
- •Useless when problem is hard to reproduce, or occurs on server in unattended operation, or at remote user site.

(21)

Static tracing

Three approaches:
- •Roll your own code.
- •Simple Debug class.
- •Log4J, java.util.logging: industrial strength

(22)

Roll-your-own tracing

Use a DEBUG constant and if statement to toggle tracing on and off.

Advantages:
- •Execution efficiency
- •Debugging code compiled out when DEBUG is false.

Disadvantages:
- •Inflexible, adds code mass.
- •Debugging code compiled out when DEBUG is false.

(23)

Roll-your-own example

Assume that expensiveMethod() is very costly in time and resources to compute. What is the cost of this debugging statement when DEBUG is false?

```
Class Foo {
  private static final boolean DEBUG=false;
  :
  public void doSomething() {
    if (DEBUG) {
      System.out.println(expensiveMethod());
    }
  }
}
```

(24)

Tracing support: Debug Class

Advantages over hand-coded tracing:
- Trace statements organized into groups.
- Automatic time-stamping of trace output
- Run-time configuration of which trace groups to print.

Disadvantages:
- Trace statements cannot be compiled out.
- No hierarchy of tracing groups.
- Hardcoding of trace group names.
- Inefficiency resulting from string concatenation.
- Trace log output cannot be redirected.

See Stack module in CVS for code example.

(25)

Debug tracing example

Assume that expensiveMethod() is very costly in time and resources to compute. What is the cost of this debugging statement when Debug.FOO is false?

```
Class Foo {
  public void doSomething( ) {
    Debug.println(Debug.FOO, expensiveMethod( ));
  }
}
```

(26)

Log4J: Industrial Tracing

Overcomes disadvantages of Debug class:
- Output can be redirected to one or more streams or even to remote sites.
- Execution efficiency improved.
- Trace group hierarchies available.
- Logging can be controlled after shipping by editing a configuration file without touching binary.
- Ported to Python, C++, C#

Disadvantage:
- somewhat complicated to learn.
- Java 1.4 includes java.util.logging API.

(27)

Dynamic Tracing

Observe program in real time as it executes.

Basic steps:
- Prepare program for dynamic tracing (-g)
- Run program under dynamic trace facility (jdb, built-ins for Jbuilder, etc.)
- Control execution (setting breakpoints, stepping, etc.)

(28)

Preparation for dynamic tracing

Compile classes with –g option
- Lets you examine local, class instance, and static variables when debugging.
- Some of the core Java platform system classes are not compiled with –g, so their local variables etc are not available to you unless you recompile them as well.

Must find out how to do this for each IDE; read the documentation.

(29)

Setting breakpoints

Breakpoints:
- Temporary markers you place in your program to tell the debugger where to stop program execution.
- Onec stopped, you can check the contents of variables, registers, storage, and stack.

Breakpoint types:
- Line breakpoints: triggered before a line of code.
- Method breakpoints: triggered before a method is invoked.
- Counter breakpoints: triggered before a counter reaches a certain value.
- Exception breakpoints: triggered when an exception of a certain type is raised.
- Storage change breakpoints: triggered when a storage location has its value changed.
- Address breakpoints: triggered when a specific address in memory is reached.

(30)

Stepping

Once a breakpoint is reached, you continue execution via "stepping":

Stepping into:
- Executes current line. If line contains a method call, debugger traverses into method and stops.

Stepping over:
- Executes current line and stops at next line in current method (does not stop inside current line.)

Step return:
- Continues execution until caller of current line is returned to (stack pop)

(31)

JDB

Provided with Sun's java distribution.

Features:
- Free
- Platform independent
- Runs as separate process

Downsides:
- Primitive command-line interface.

(32)

IDE Example: JBuilder

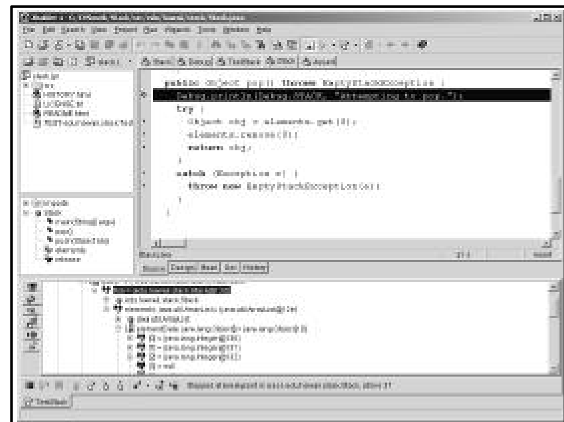
JBuilder: debug info compiled in by default
- Project Properties|Build|Include Debug Info
- (Ant: javac task must have debug="on")

Set breakpoints by clicking in margin

Run under debugger
- Run|Debug Project

Tutorial available in Jbuilder documentation.

(33)



Miscellaneous issues

Exceptions

Debugging by ablation

Bug report guidelines

Domain specific debugging

First and last resorts

(35)

Design of Exceptions

Exceptions should provide information on:
- Type of error (Exception type)
- Where it occurred (stack trace)
- Useful additional data (message)

Don't swallow exceptions:
- `catch (exception e) { }`

Don't throw everything
- If a method throws more than three types of exceptions, it often indicates poor design.

(36)

Debugging by Ablation

Create the smallest possible program that still exhibits the problem and debug that.

(37)

Bug Report Guidelines

The prime directive: Reproducibility

- If a bug report is not detailed enough to allow the developer to reproduce it, it won't generally be worked on.

Details that support reproducibility:

- System
- Version
- Component
- Platform
- OS
- Severity
- Steps to reproduce
- Expected results, and Actual results

(38)

Domain specific debugging

There are additional techniques and technologies that can be applied for debugging certain application domains:

- Enterprise JavaBean systems
- Servlets
- RMI (distributed applications)
- Multi-threaded applications
- Performance debugging
  - Memory leaks
  - Time/space optimization

(39)

First and last resorts

Desk checking:

- Print out the source code and read it.
- Invent some test data and mentally follow its execution.

Walk through:

- Explain your code to someone else.

Mental cold reboot:

- Sleep on it.
- Get some exercise.

Post a question to a newsgroup.

- But know how to write a post that will elicit a helpful response!
  - http://www.tuxedo.org/~esr/faqs/smart-questions.html
  - http://www.chiark.greenend.org.uk/~sgtatham/bugs.html

(40)

Relative Efficiencies

Extreme Debugging:

- Most efficient approach
- Automated error detection
- Effort of test case development amortized
- Benefits developers other than test author.

Static Tracing:

- Relatively efficient
- Manual error detection
- Effort of static trace development amortized
- Benefits developers other than trace author.

Dynamic Tracing:

- Least efficient approach.
- Manual error detection.
- No amortization of dynamic trace effort
- No benefits to developers other than trace author.

(41)