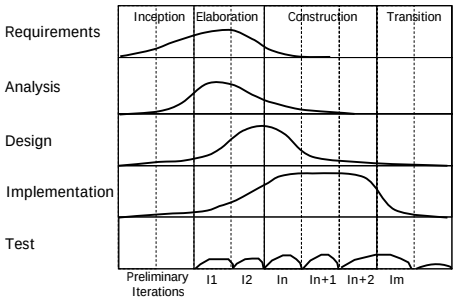
UCL

Debuggers

Wolfgang Emmerich
Professor of Distributed Computing
University College London
http://sse.cs.ucl.ac.uk

UCL

Context



2

UCL

Learning Objectives

- To appreciate the tool support that debuggers make available for understanding the cause of defects
- To understand the basic requirements that programmers have for debuggers
- To be able to use the debugging functionality that modern debuggers provide effectively
- To be aware that it is possible to extend debugging functionality to non-traditional languages (e.g. BPEL)

3

UCL

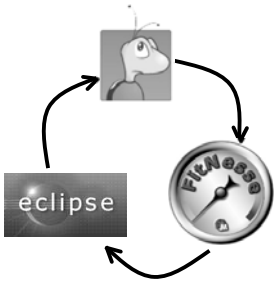
What is a debugger?

- A *debugger* is a tool that supports programmers in the task of understanding the cause of defects in computer programs
- Debuggers are a complementary to, and not a replacement for, testing tools
- Used to be sniffed at by (theoretical) computer scientists who argued that they are unnecessary because programs should be correct from the start
- Instead debuggers are crucial tools for detecting the cause of defects in a cost-effective manner

4

UCL

Bug tracking, testing tools and debuggers



- Defects get lodged in bug or issue management tools (e.g. bugzilla)
- Next use testing tools (e.g. FitNesse) to add a test case to your test suite to detect and reproduce the defect
- In order to understand why that defect appears you use a debugger
- You might find further bugs while doing so and you then lodge them in the bug tracker, write tests, …

5

UCL

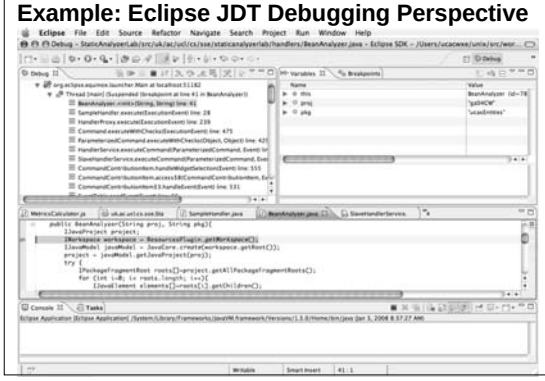
Requirements for debuggers

- Tools to “open the program black-box”
- Check that program instructions have the desired effect
- To do this need to be able to:
  - Break / resume the execution
  - Execute a program step-by-step
  - Introspect the program state
    - Object state,
    - Local variable and parameter values
    - Thread structure
    - Call stack
  - Evaluate expressions on the fly
  - Trace program execution
- At source-code abstraction level

6

UCL

Example: Eclipse JDT Debugging Perspective



7

UCL

Breakpoints

- A breakpoint determines when program execution should be suspended
- Manage more than one breakpoint
  - Enable breakpoints
  - Suspend breakpoints
- Conditional breakpoints
  - Only stop when a condition is true
  - Only stop when an exception is raised

8

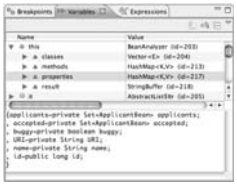
UCL

Stepping through a program

- Once execution has been suspended programmers want to step through code
- Cursor highlights current program position
- Stepping into/over/out from method calls
- Cursor changes to next statement that will be executed
- Continue execution until the next breakpoint

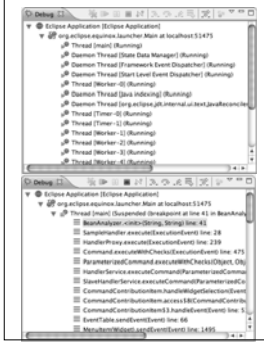
9

Introspection of objects and variables



- Check which fields constitute the state of objects
- Check the value of
  - fields
  - local variables
  - parameters
- Display value of expressions
- Watch how the value of expressions changes over time

Introspection of threads and call stack



- Important to understand how many threads there are
- Need to know which thread is doing what
- Can set breakpoints and once a thread reaches one it will be suspended
- Then debugger should allow introspection of the call stack
- Helps to ascertain correctness of call graph
- Navigation to source code using the call graph information

Building debuggers

- Debugging support is part of the Eclipse platform
- Offers language independent debugging support:
  - Ability to lauch processes
  - Concepts of breakpoints
  - Correlate processes and source code
  - Editor / debugger interactions
- Language-specific debuggers use extension mechanisms to provide language-specific support, e.g.
  - Java debugger in JDT
  - C++ debugger in CDT
- Debuggers are tools of substantial size
  - Eclipse Debugging Platform has 146kLoC
  - Eclipse JDT Debugger has 130kLoC

UCL

Key Points

- Debuggers support programmers in finding the cause of defects by:
  - Traversal through program execution and
  - Introspection of program state
- Modern debuggers are symbolic and work at source-code level
- To achieve this debuggers are integrated into IDEs

13