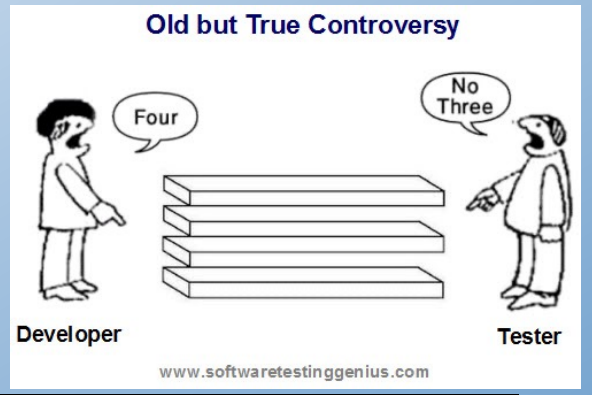


Testing II

CSCI2340: Software Engineering of Large Systems

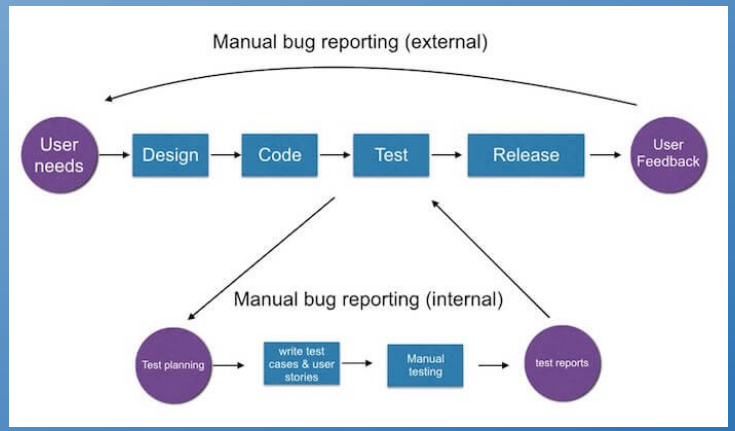
Steven P. Reiss

Creating Test Cases

- White box vs Black box testing
  - Black box: only the interface is known
    - Tester must assume something about the code
  - White box: code is available
    - Tester can read the code to find potential problems
    - Much easier and more practical
- Basic Test Cases (programmer created)
  - Test basic functionality to see if program works
    - Needed to assist development
    - Can run through typical scenarios
    - Test extended functionality as it is added
    - Make these permanent
  - Tests to ensure the program can handle bad inputs
  - Tests to catch possible errors
    - A successful test case finds a bug
  - These are only a small fraction of a test suite

Creating Test Cases

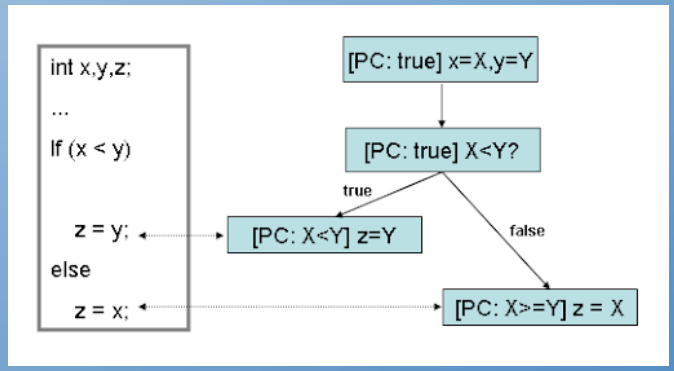
- Each user reported bug should have a test case
  - Unless the bug and fix are trivial
  - To duplicate & fix the bug (and check the fix works)
  - To prevent regression
  - Most tests in a suite are of this type
  - These can be from actual bug reports
  - Or from automatically recorded stack traces
- Creating test cases to ensure complete coverage
  - You want to ensure all code is tested
  - Based on type of coverage desired
  - Additional tests are generally needed to achieve this



Creating Test Cases

- Creating test cases is a lot of work
  - Finding appropriate arguments to get coverage
  - Finding arguments that might cause the system to fail
  - Finding arguments that duplicate a bug report or stack trace
  - Setting up the appropriate environments
  - Checking the results
  - Programmers get sloppy (tests are buggy, not the code)
- This leads to work on automatic test generation
  - With a variety of different approaches
  - Commercial and research-based systems exist
  - LLMs (e.g., Claude, ChatGPT) can do some of this (but not all)

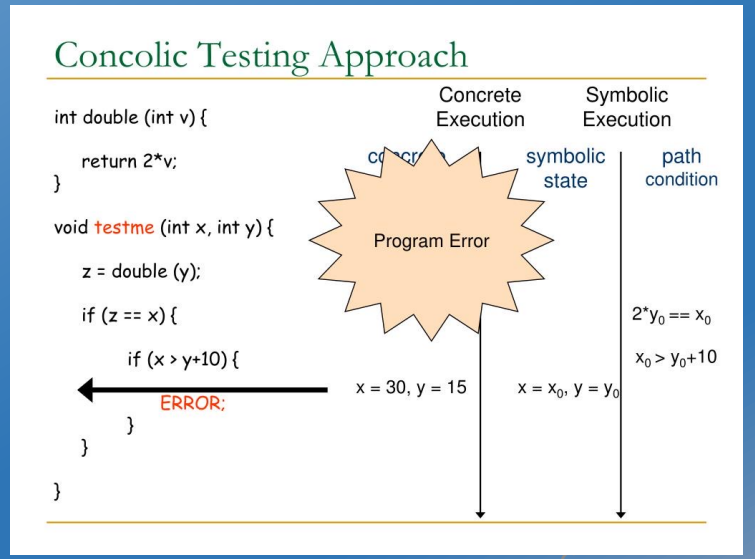
Symbolic Execution



- For each coverage point (line/branch/condition)
  - Determine constraints on variables needed to get there
  - Trace constraints back to the start of the function
  - This gives you constraints on the test inputs
- Construct new inputs based on these constraints
- Often difficult to do
  - The problem in general is unsolvable (halting problem)
  - Works best with numerical routines and inputs
  - Can use shape analysis for objects
  - Often fails

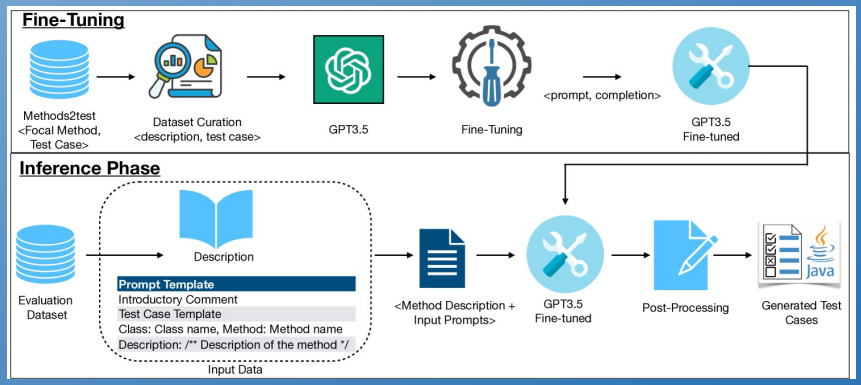
Concolic Testing

- Combination of concrete testing and symbolic execution
- Do symbolic execution along a concrete execution path
  - Easiest when starting with a failed execution
    - Given values at that point: compute values at the start of the function
  - Forward analysis
  - Maintaining constraint expressions for each variable
- More practical than pure symbol execution
  - But not as complete
  - And not necessarily more useful
- Test generation tools
  - EvoSuite, qodo, …
    - Work okay, but not great
  - But is this what you want?
    - What is the test output; what is successful?
    - Is the test an important one?

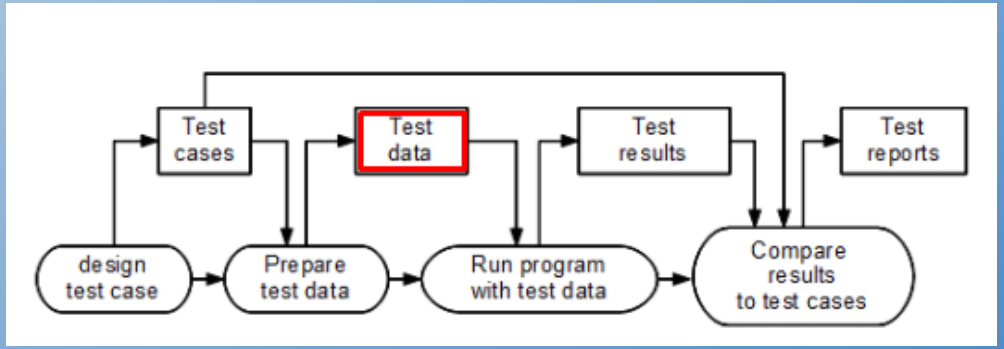


Using LLMs to Create Test Cases

- You can ask a LLM to create test cases for a method
  - It can do a reasonable job of creating basic tests
    - Normal functionality
    - Edge cases
  - BUT
    - It can not achieve strong coverage
    - It can provide the wrong outputs
    - It is more geared to method rather than system testing
- This will improve in the future
  - But still probably won't be complete



Creating Tests for a Bug



- This is a standard part of debugging
  - Much of a test suites is developed this way
- Research: how to automate this
  - If a bug report has a stack trace
  - If the bug report has enough information
  - Using concolic testing techniques
  - From a debugger situation
- ROSE test generation (using debugger & input)
  - ROSE is our automatic program repair tool
  - ROSE knows the failure symptom and the environment
    - Identifies what portions of the environment are needed for the test
    - Environment can be queried via the debugger at the time
  - Should be able to generate a test case based on this
  - Difficulty is recreating the environment using only accessible methods
  - Might not know the correct result (other than no exception)

Testing Interactive Programs

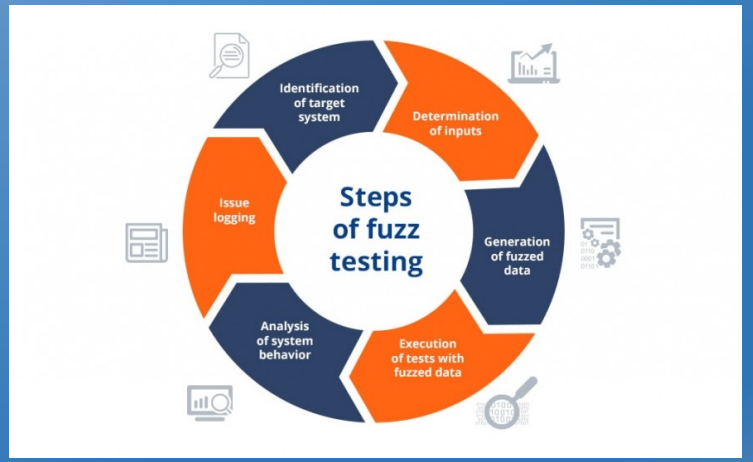
- Problem
  - The test is a sequence of interactions
  - You don’t want to have a user do the interactions
    - Although this is done
  - You don’t want to redo the test if the UI is rearranged
    - Different window sizes, visual updates, …
- Various solutions exist
  - Generating RESTful calls without a UI (curl)
  - Tracking and repeating user interactions
  - Writing code to emulate a user using widget accessors

Tools for Interactive Testing

- Tools exist to help with interactive testing
- Selenium is the one I've seen used the most
  - Designed for web pages and the browser
    - Appium/Selendroid extend it for mobile devices
    - Similar tools exist for desktop applications
  - Can record a sequence of user interactions
    - Generates a program that identifies widgets by CSS accessor
  - User can write programs to emulate the interactions
    - Starting with recorded or separately
    - Separate functions for common interactions
  - Result is a testing framework for the application
- Flutter/Dart: built in tool + several alternatives
- Other tools: Squish, Sikuli, …
- Research: generating tests for RESTful calls based on front end

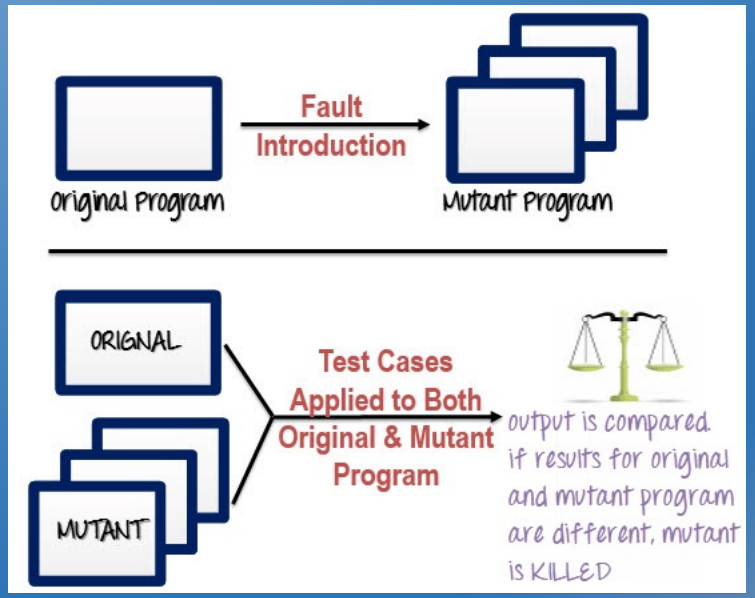
Fuzz Testing

- Another approach to automatic test generation
- Generate invalid, unexpected or random inputs to a routine
  - Try out lots of values to get a function to fail
  - Generate tests where the function fails
- Like letting naïve users try the software
  - Cat on the keyboard
  - Freshmen attacking FIELD
  - 5-year-old at the keyboard

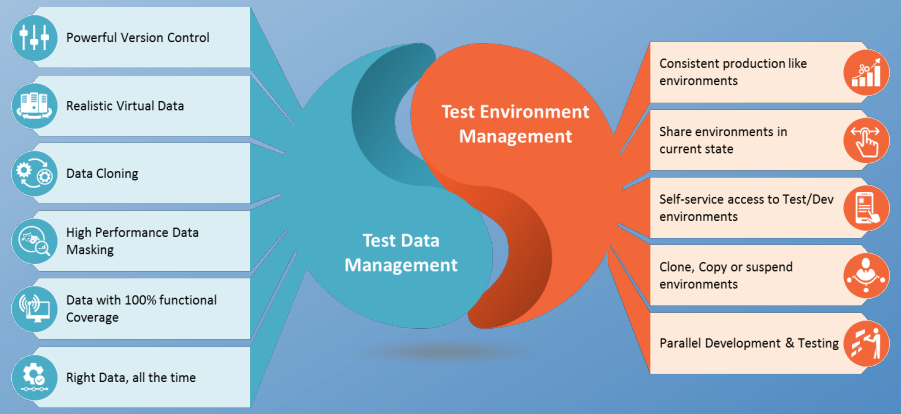


Mutation Testing

- Change the code slightly (semi-random mutations)
  - Standard set of possible mutations
    - E.g., invert a conditional
  - Check if the test cases can detect the change
    - This helps evaluate the test suite
    - Note that many obvious errors are not detected
- Find test cases that can detect the change
  - This broadens the test suite



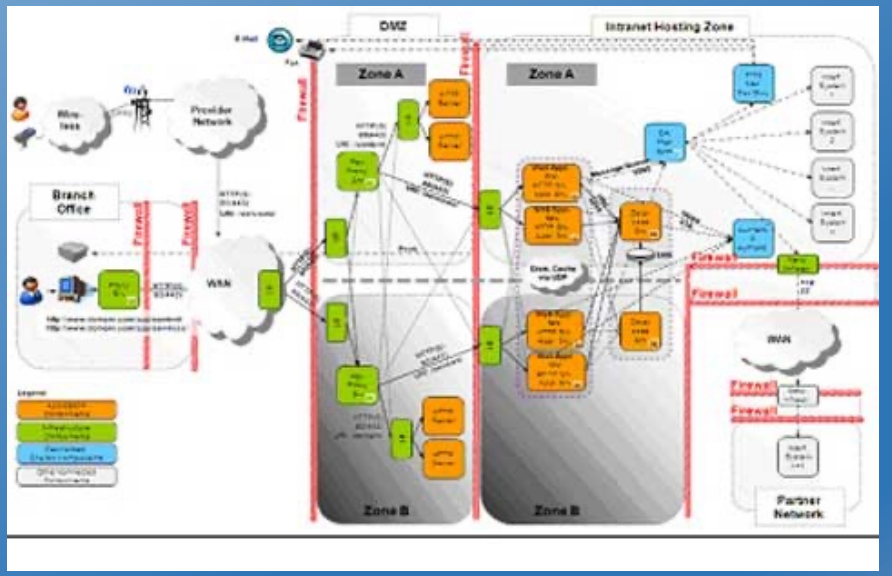
Test Environments



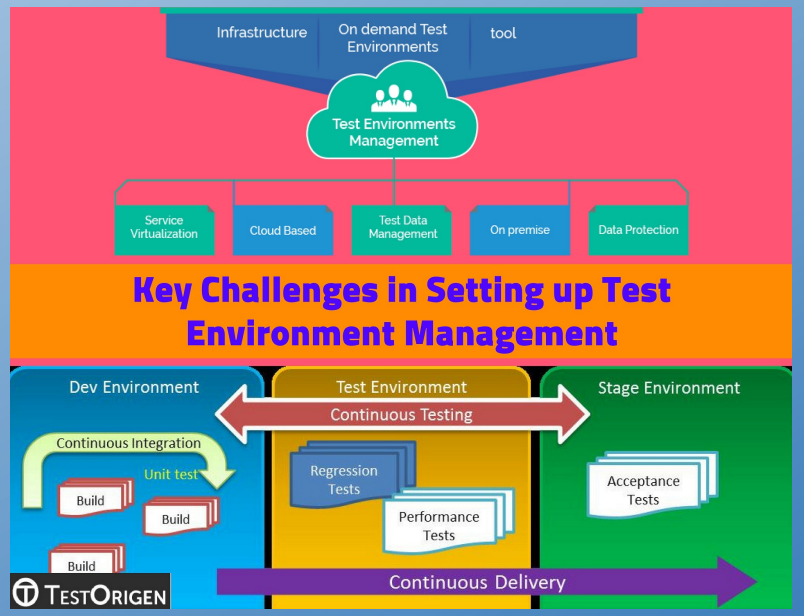
- How are you going to run your tests?
- Testing and production can be quite different
  - Production database with real people, data
  - Has external effects
    - Charging credit cards, sending data to warehouse, causes things to be shipped
  - Accumulating usage data for business purposes
  - Requiring external web browser or talks to real hardware
  - Changing files in the file system
- All these might not work in a test environment
  - And you don't want to test new code in production
  - Production code needs to be tested and robust
  - Can't afford to break hardware while testing

Test Environment Contents

- Separate git branch from the production system
  - Or just don't pull onto production host until version is stable
    - And test from current version on another host
  - Or create the branch when production is ready (it will still change)
    - This is what is typically done in active systems
  - Or run/compile time flags
  - Or detect what machine it is running on (or who is running it)
- Separate database
  - Preferred over special items in production database
- Separate hooks for external features
  - Payments, real-world consequences
  - Simulation of outside hardware
  - Simulation of the real world (outside, real-time events)
- Might want to cache outside queries
- Separate machine to run on

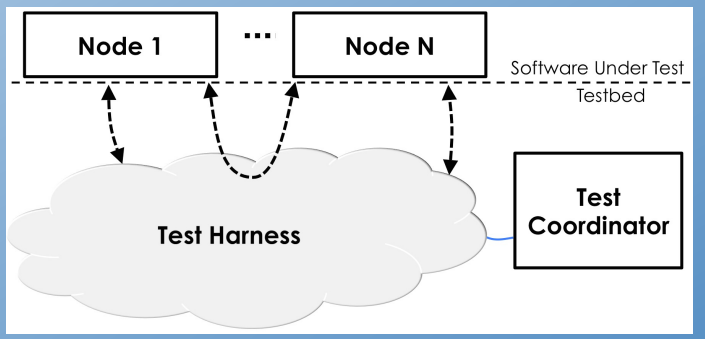


Setting up a Test Environment



- Should have a script to do this
  - Restore the test file system to a known state
  - Restore test database to a known state
  - Set up any other environmental data
  - Even if you only use it for running a test suite
- Some can be done with the individual tests
  - Using @Before, @After, @BeforeClass, @AfterClass
  - Not ideal for file system, database setup
- Handling multiple machines and processes
  - Might require test harness or coordinator

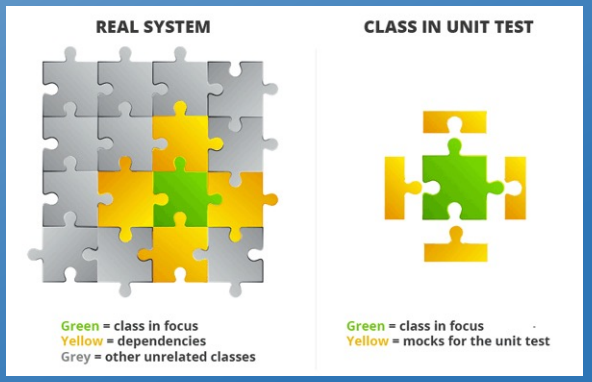
Testing System Components



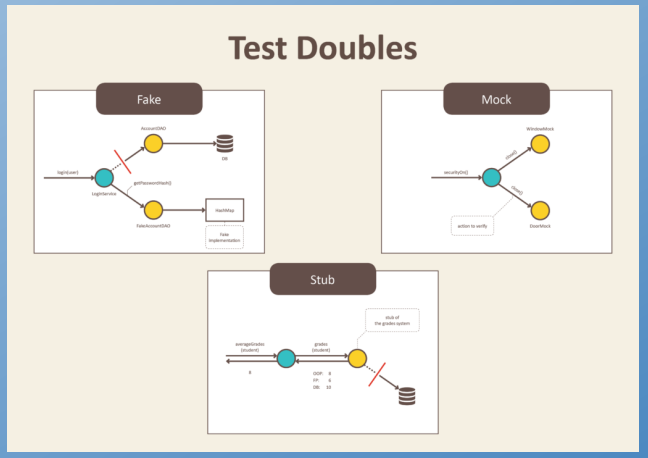
- Want to check calls to other components are right, but not have a real effect
  - Want to test a web back end without using a browser
  - Want to test web front end without using the back end
  - Want to test payment without paying, shopping without buying
  - Want to test SHORE without trains
  - Want to test Pinball without the pinball machine
- Rather than trying to use actual code
  - Create code that mimics the actual code
  - Might be simple (accept a particular card number, reject others)
  - Might check that behavior of the system is correct
  - This is called **MOCKING**
  - Same issue when system pieces are missing
  - Some external libraries do this for you already (Stripe, for example)
- Simulation is another alternative
  - Especially for hardware systems and real-world interaction
  - Simulated pinball machine; simulated smart house

Mocking

- Mock component is generally much simpler than actual one
- Provide reasonable returns, but generally fixed return values
- Provide hooks to handle the whole interface
  - With minimal responses unless needed for testing
  - Provide minimum functionality needed to support tests
- Methods that take inputs should check their values
  - This is part of testing
- Providing logging for additional test validation
- Frameworks exist to help create mocking code
  - Mockito is the standard for Java
  - Easier, but not necessarily recommended in the long term



Mocking

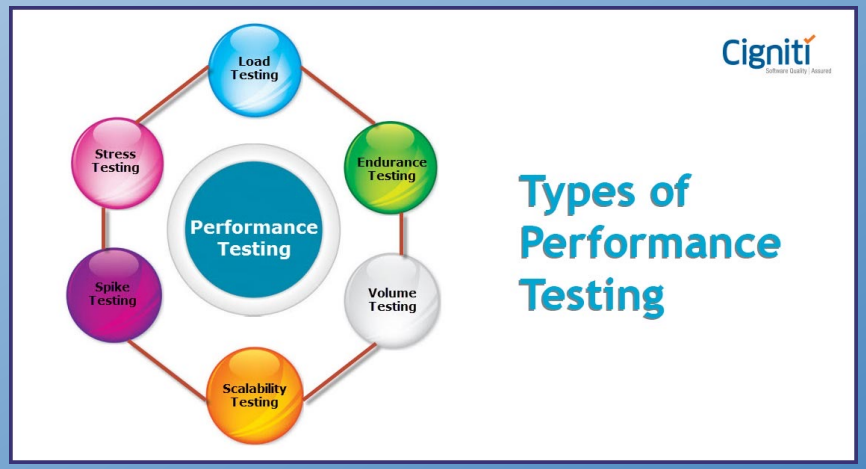


- Mocking code is not throw-away
  - Especially mocking for testing
  - Take care in writing it as you would system code
  - Buggy mocking code will cause bad testing
    - And it is likely to be buggy
  - Use established coding practices
- Might want to create mocking code for development
  - For portions of the system not yet implemented

Security Testing

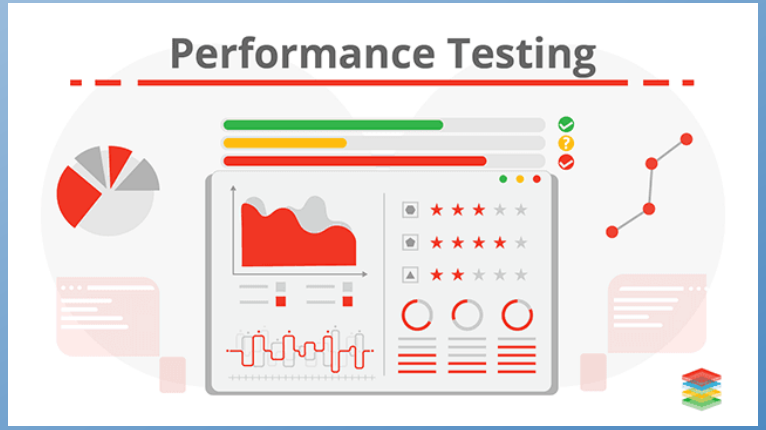
- No system is totally secure
  - Need to test for potential security problems
- Checking for known security problems
  - C/C++: checking for buffer overflow possibilities
  - Web Apps: SQL injection, XSS, DoS and other attacks
- Dynamic security checking tools exist (usually web-oriented)
  - Machine vulnerabilities
  - SQL injection attacks, Cross-site scripting, Denial of Service, server configuration
- You should think about applying these to your system
  - Ethical hacking
- Static security checking tools exist (verification) is often done in addition to testing
  - Partial correctness formal methods
    - Checking buffer overflows
    - Checking tainted data
    - Checking program states
  - Since you want to check ALL inputs

Performance Testing



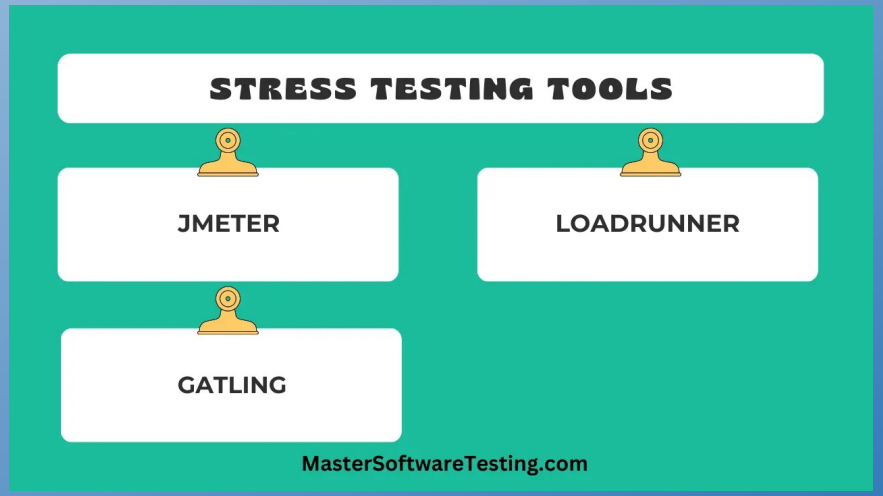
- Performance can be important
  - Resource utilization (CPU and memory)
  - User perception
    - 100ms makes a noticeable difference in usability
  - Ability to handle large data, large loads, complex cases, …
- Stress testing
  - What happens if files are large (user uploads 100G file)
  - What happens if you have 100,000 users
  - How does the system degrade (or does it crash)

Performance Testing



- Typically, performance doesn't matter
  - 10% of the code uses 90% of the time (5/95)
  - Most of that 10% is outside of your control
    - Necessary functionality
    - Inherent to the application
- Performance only matters if there is a problem
  - User interface is too slow
  - Using too many resources (memory, CPU, disk, …)
  - Otherwise, simplicity and ease of coding reign

Detecting Performance Problems

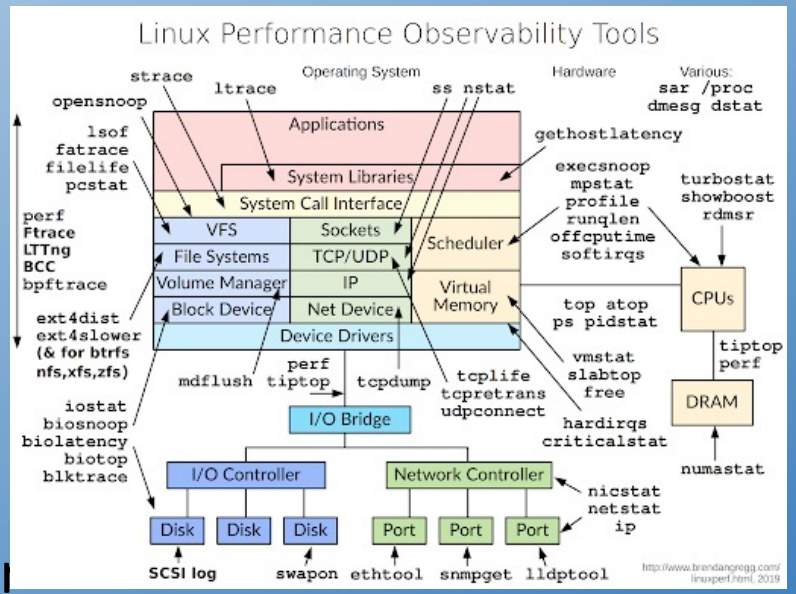


- User interaction performance
  - What users perceive (what you perceive)
  - You can get timings (video and look at frames)
    - Browser timings for web applications
- Application performance
  - Time for specific operations
  - System view of CPU, memory, disk I/O
- Multithreaded performance
  - % CPU used vs expected (improvement from multiple threads)
  - Tools to understand behavior: locking, bottlenecks, …
- Stress testing
  - Simulate the load: jmeter and similar tools

Creating Performance Test Cases

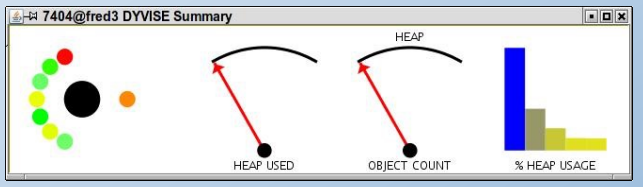
- Normal test case that takes too long
- Typically, not done as part of test suite
  - Can be done to debug the performance problem
    - Create a test for this purpose
    - Might not be considered part of test suite
  - Can have a test case that times out to fail
    - But running on a different machine …
  - Might be non-deterministic
    - You might need to run it a lot of times

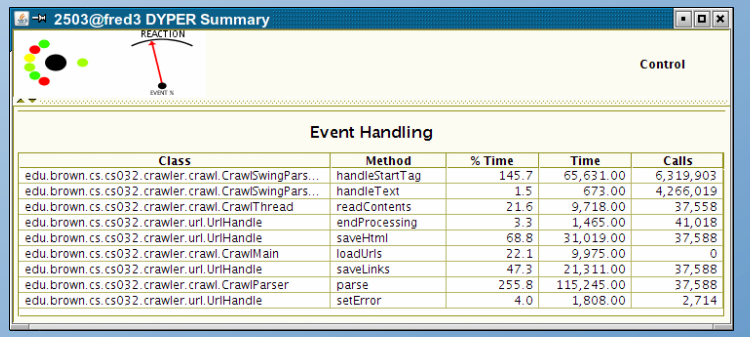
Performance Analysis Tools

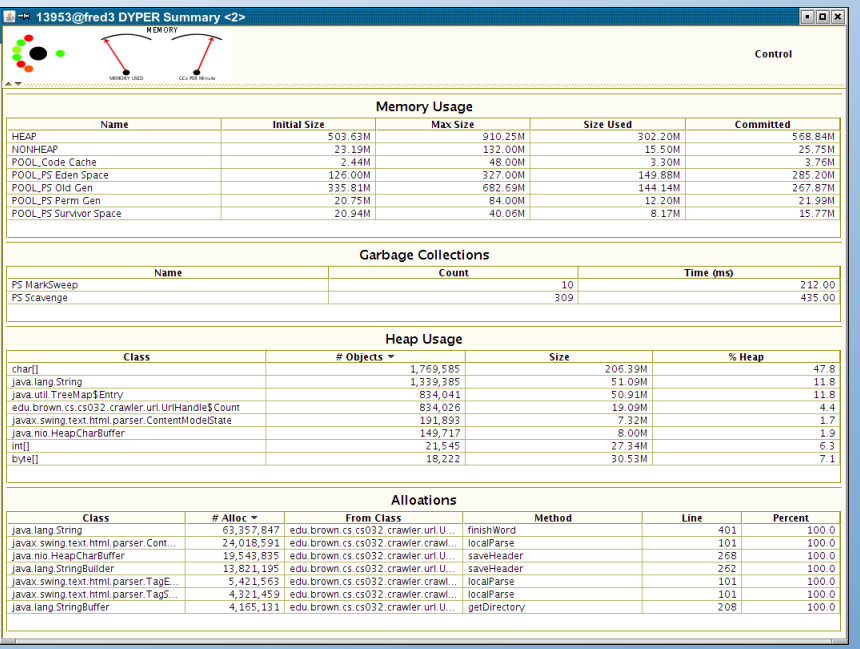


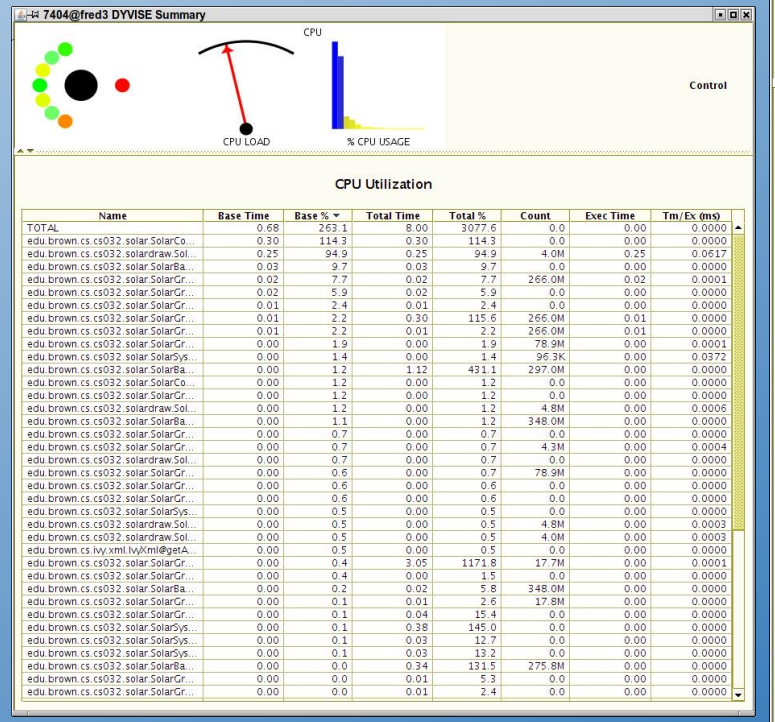
- UNIX / LINUX: prof, gprof
  - For C/C++ and native code
  - prof gives time & count for each function
  - gprof gives these for each function + caller-callee pair
    - Get an approximation to why a routine spends its time
- TypeScript
  - Chrome DevTools
- Standalone Java tools
  - Jprofiler, VisualVM, jconsole
  - Dymon
- These provide limited information
  - Especially about multithreaded problems

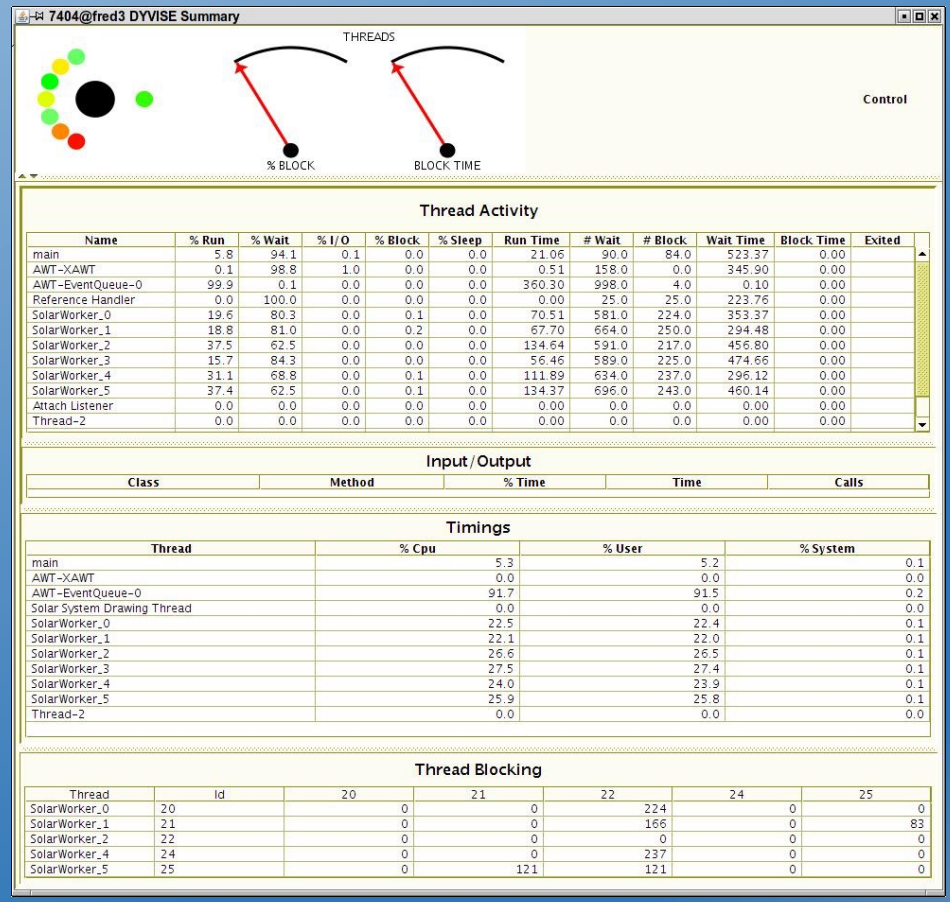
DYMON


7404@fred3 DYVISE Summary
HEAP
HEAP USED
OBJECT COUNT
% HEAP USAGE


2503@fred3 DYPER Summary
REACTION
Control
Event Handling
Class | Method | % Time | Time | Calls
edu.brown.cs.cs032.crawler.crawl.CrawlSwingPars... | handleStartTag | 145.7 | 65,631.00 | 6,319,903
edu.brown.cs.cs032.crawler.crawl.CrawlSwingPars... | handleText | 1.5 | 673.00 | 4,266,019
edu.brown.cs.cs032.crawler.crawl.CrawlThread | readContents | 21.6 | 9,718.00 | 37,558
edu.brown.cs.cs032.crawler.url.UrlHandle | endProcessing | 3.3 | 1,465.00 | 41,018
edu.brown.cs.cs032.crawler.url.UrlHandle | saveHtml | 68.8 | 31,019.00 | 37,588
edu.brown.cs.cs032.crawler.crawl.CrawlMain | loadUrls | 22.1 | 9,975.00 | 0
edu.brown.cs.cs032.crawler.url.UrlHandle | saveLinks | 47.3 | 21,311.00 | 37,588
edu.brown.cs.cs032.crawler.crawl.CrawlParser | parse | 255.8 | 115,245.00 | 37,588
edu.brown.cs.cs032.crawler.url.UrlHandle | setError | 4.0 | 1,808.00 | 2,714

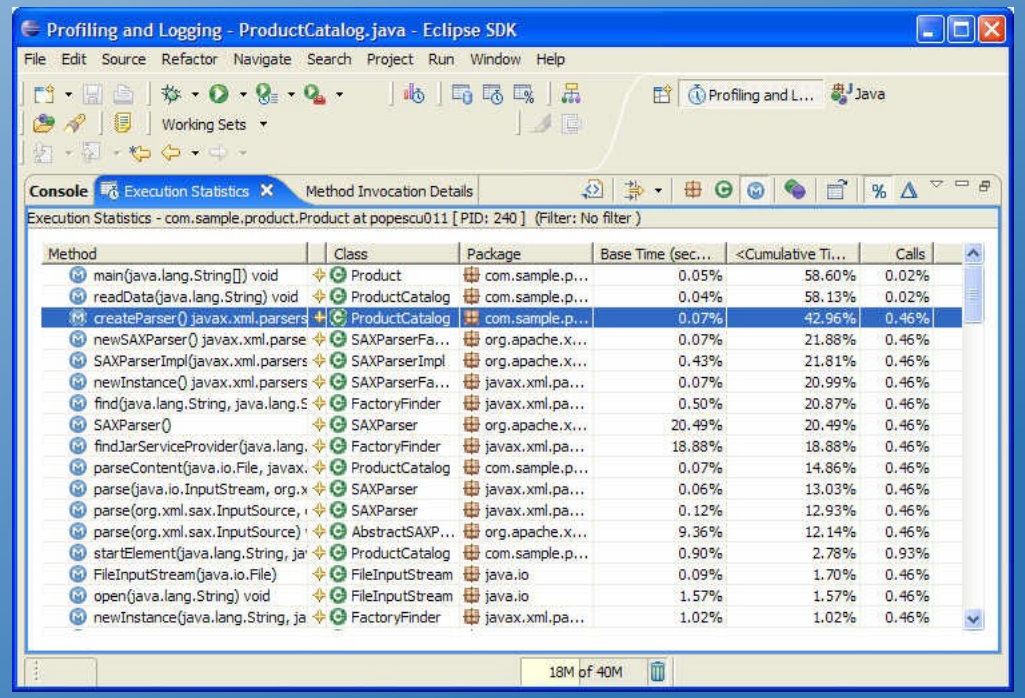

13953@fred3 DYPER Summary <2>
MEMORY
Control
Memory Usage
Name | Initial Size | Max Size | Size Used | Committed
HEAP | 503.63M | 910.25M | 302.20M | 568.84M
NONHEAP | 23.19M | 132.00M | 15.50M | 25.75M
POOL_Code Cache | 2.44M | 48.00M | 3.30M | 3.76M
POOL_PS Eden Space | 126.00M | 327.00M | 149.88M | 285.20M
POOL_PS Old Gen | 335.81M | 682.69M | 144.14M | 267.87M
POOL_PS Perm Gen | 20.75M | 84.00M | 12.20M | 21.99M
POOL_PS Survivor Space | 20.94M | 40.06M | 8.17M | 15.77M
Garbage Collections
Name | Count | Time (ms)
PS MarkSweep | 10 | 212.00
PS Scavenge | 309 | 435.00
Heap Usage
Class | # Objects | Size | % Heap
char[] | 1,769,585 | 206.39M | 47.8
java.lang.String | 1,339,385 | 51.09M | 11.8
java.util.TreeMap$Entry | 834,041 | 50.91M | 11.8
edu.brown.cs.cs032.crawler.url.UrlHandle$Count | 834,026 | 19.09M | 4.4
javax.swing.text.html.parser.ContentModelState | 191,893 | 7.32M | 1.7
java.nio.HeapCharBuffer | 149,717 | 8.00M | 1.9
int[] | 21,545 | 27.34M | 6.3
byte[] | 18,222 | 30.53M | 7.1
Allocations
Class | # Alloc | From Class | Method | Line | Percent
java.lang.String | 63,357,847 | edu.brown.cs.cs032.crawler.url.U... | finishWord | 401 | 100.0
javax.swing.text.html.parser.Cont... | 24,018,591 | edu.brown.cs.cs032.crawler.crawl... | localParse | 101 | 100.0
java.nio.HeapCharBuffer | 19,543,835 | edu.brown.cs.cs032.crawler.url.U... | saveHeader | 268 | 100.0
java.lang.StringBuilder | 13,821,195 | edu.brown.cs.cs032.crawler.url.U... | saveHeader | 262 | 100.0
javax.swing.text.html.parser.TagE... | 5,421,563 | edu.brown.cs.cs032.crawler.crawl... | localParse | 101 | 100.0
javax.swing.text.html.parser.TagS... | 4,321,459 | edu.brown.cs.cs032.crawler.crawl... | localParse | 101 | 100.0
java.lang.StringBuffer | 4,165,131 | edu.brown.cs.cs032.crawler.url.U... | getDirectory | 208 | 100.0


7404@fred3 DYVISE Summary
CPU
CPU LOAD
% CPU USAGE
Control
CPU Utilization
Name | Base Time | Base % | Total Time | Total % | Count | Exec Time | Tm/Ex (ms)
TOTAL | 0.68 | 263.1 | 8.00 | 3077.6 | 0.0 | 0.00 | 0.0000
edu.brown.cs.cs032.solar.SolarCo... | 0.30 | 114.3 | 0.30 | 114.3 | 0.0 | 0.00 | 0.0000
edu.brown.cs.cs032.solardraw.Sol... | 0.25 | 94.9 | 0.25 | 94.9 | 4.0M | 0.25 | 0.0617
edu.brown.cs.cs032.solar.SolarBa... | 0.03 | 9.7 | 0.03 | 9.7 | 0.0 | 0.00 | 0.0000
edu.brown.cs.cs032.solar.SolarGr... | 0.02 | 7.7 | 0.02 | 7.7 | 266.0M | 0.02 | 0.0001
edu.brown.cs.cs032.solar.SolarGr... | 0.02 | 5.9 | 0.02 | 5.9 | 0.0 | 0.00 | 0.0000
edu.brown.cs.cs032.solar.SolarGr... | 0.01 | 2.4 | 0.01 | 2.4 | 0.0 | 0.00 | 0.0000
edu.brown.cs.cs032.solar.SolarGr... | 0.01 | 2.2 | 0.30 | 115.6 | 266.0M | 0.01 | 0.0000
edu.brown.cs.cs032.solar.SolarGr... | 0.01 | 2.2 | 0.01 | 2.2 | 266.0M | 0.01 | 0.0000
edu.brown.cs.cs032.solar.SolarGr... | 0.00 | 1.9 | 0.00 | 1.9 | 78.9M | 0.00 | 0.0001
edu.brown.cs.cs032.solar.SolarSys... | 0.00 | 1.4 | 0.00 | 1.4 | 96.3K | 0.00 | 0.0372
edu.brown.cs.cs032.solar.SolarBa... | 0.00 | 1.2 | 1.12 | 431.1 | 297.0M | 0.00 | 0.0000
edu.brown.cs.cs032.solar.SolarCo... | 0.00 | 1.2 | 0.00 | 1.2 | 0.0 | 0.00 | 0.0000
edu.brown.cs.cs032.solar.SolarGr... | 0.00 | 1.2 | 0.00 | 1.2 | 0.0 | 0.00 | 0.0000
edu.brown.cs.cs032.solardraw.Sol... | 0.00 | 1.2 | 0.00 | 1.2 | 4.8M | 0.00 | 0.0006
edu.brown.cs.cs032.solar.SolarBa... | 0.00 | 1.1 | 0.00 | 1.2 | 348.0M | 0.00 | 0.0000
edu.brown.cs.cs032.solar.SolarGr... | 0.00 | 0.7 | 0.00 | 0.7 | 0.0 | 0.00 | 0.0000
edu.brown.cs.cs032.solar.SolarGr... | 0.00 | 0.7 | 0.00 | 0.7 | 4.3M | 0.00 | 0.0004
edu.brown.cs.cs032.solardraw.Sol... | 0.00 | 0.7 | 0.00 | 0.7 | 0.0 | 0.00 | 0.0000
edu.brown.cs.cs032.solar.SolarGr... | 0.00 | 0.6 | 0.00 | 0.7 | 78.9M | 0.00 | 0.0000
edu.brown.cs.cs032.solar.SolarGr... | 0.00 | 0.6 | 0.00 | 0.6 | 0.0 | 0.00 | 0.0000
edu.brown.cs.cs032.solar.SolarGr... | 0.00 | 0.6 | 0.00 | 0.6 | 0.0 | 0.00 | 0.0000
edu.brown.cs.cs032.solar.SolarSys... | 0.00 | 0.5 | 0.00 | 0.5 | 0.0 | 0.00 | 0.0000
edu.brown.cs.cs032.solardraw.Sol... | 0.00 | 0.5 | 0.00 | 0.5 | 4.8M | 0.00 | 0.0003
edu.brown.cs.cs032.solardraw.Sol... | 0.00 | 0.5 | 0.00 | 0.5 | 4.0M | 0.00 | 0.0003
edu.brown.cs.ivy.xml.IvyXml@getA... | 0.00 | 0.5 | 0.00 | 0.5 | 0.0 | 0.00 | 0.0000
edu.brown.cs.cs032.solar.SolarGr... | 0.00 | 0.4 | 3.05 | 1171.8 | 17.7M | 0.00 | 0.0001
edu.brown.cs.cs032.solar.SolarGr... | 0.00 | 0.4 | 0.00 | 1.5 | 0.0 | 0.00 | 0.0000
edu.brown.cs.cs032.solar.SolarBa... | 0.00 | 0.2 | 0.02 | 5.8 | 348.0M | 0.00 | 0.0000
edu.brown.cs.cs032.solar.SolarGr... | 0.00 | 0.1 | 0.01 | 2.6 | 17.8M | 0.00 | 0.0000
edu.brown.cs.cs032.solar.SolarGr... | 0.00 | 0.1 | 0.04 | 15.4 | 0.0 | 0.00 | 0.0000
edu.brown.cs.cs032.solar.SolarSys... | 0.00 | 0.1 | 0.38 | 145.0 | 0.0 | 0.00 | 0.0000
edu.brown.cs.cs032.solar.SolarSys... | 0.00 | 0.1 | 0.03 | 12.7 | 0.0 | 0.00 | 0.0000
edu.brown.cs.cs032.solar.SolarSys... | 0.00 | 0.1 | 0.03 | 13.2 | 0.0 | 0.00 | 0.0000
edu.brown.cs.cs032.solar.SolarBa... | 0.00 | 0.0 | 0.34 | 131.5 | 275.8M | 0.00 | 0.0000
edu.brown.cs.cs032.solar.SolarGr... | 0.00 | 0.0 | 0.01 | 5.3 | 0.0 | 0.00 | 0.0000
edu.brown.cs.cs032.solar.SolarGr... | 0.00 | 0.0 | 0.01 | 2.4 | 0.0 | 0.00 | 0.0000


7404@fred3 DYVISE Summary
THREADS
% BLOCK
BLOCK TIME
Control
Thread Activity
Name | % Run | % Wait | % I/O | % Block | % Sleep | Run Time | # Wait | # Block | Wait Time | Block Time | Exited
main | 5.8 | 94.1 | 0.1 | 0.0 | 0.0 | 21.06 | 90.0 | 84.0 | 523.37 | 0.00
AWT-XAWT | 0.1 | 98.8 | 1.0 | 0.0 | 0.0 | 0.51 | 158.0 | 0.0 | 345.90 | 0.00
AWT-EventQueue-0 | 99.9 | 0.1 | 0.0 | 0.0 | 0.0 | 360.30 | 998.0 | 4.0 | 0.10 | 0.00
Reference Handler | 0.0 | 100.0 | 0.0 | 0.0 | 0.0 | 0.00 | 25.0 | 25.0 | 223.76 | 0.00
SolarWorker_0 | 19.6 | 80.3 | 0.0 | 0.1 | 0.0 | 70.51 | 581.0 | 224.0 | 353.37 | 0.00
SolarWorker_1 | 18.8 | 81.0 | 0.0 | 0.2 | 0.0 | 67.70 | 664.0 | 250.0 | 294.48 | 0.00
SolarWorker_2 | 37.5 | 62.5 | 0.0 | 0.0 | 0.0 | 134.64 | 591.0 | 217.0 | 456.80 | 0.00
SolarWorker_3 | 15.7 | 84.3 | 0.0 | 0.0 | 0.0 | 56.46 | 589.0 | 225.0 | 474.66 | 0.00
SolarWorker_4 | 31.1 | 68.8 | 0.0 | 0.1 | 0.0 | 111.89 | 634.0 | 237.0 | 296.12 | 0.00
SolarWorker_5 | 37.4 | 62.5 | 0.0 | 0.1 | 0.0 | 134.37 | 696.0 | 243.0 | 460.14 | 0.00
Attach Listener | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.00 | 0.0 | 0.0 | 0.00 | 0.00
Thread-2 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.00 | 0.0 | 0.0 | 0.00 | 0.00
Input/Output
Class | Method | % Time | Time | Calls
Timings
Thread | % Cpu | % User | % System
main | 5.3 | 5.2 | 0.1
AWT-XAWT | 0.0 | 0.0 | 0.0
AWT-EventQueue-0 | 91.7 | 91.5 | 0.2
Solar System Drawing Thread | 0.0 | 0.0 | 0.0
SolarWorker_0 | 22.5 | 22.4 | 0.1
SolarWorker_1 | 22.1 | 22.0 | 0.1
SolarWorker_2 | 26.6 | 26.5 | 0.1
SolarWorker_3 | 27.5 | 27.4 | 0.1
SolarWorker_4 | 24.0 | 23.9 | 0.1
SolarWorker_5 | 25.9 | 25.8 | 0.1
Thread-2 | 0.0 | 0.0 | 0.0
Thread Blocking
Thread | Id | 20 | 21 | 22 | 24 | 25
SolarWorker_0 | 20 | 0 | 0 | 224 | 0 | 0
SolarWorker_1 | 21 | 0 | 0 | 166 | 0 | 83
SolarWorker_2 | 22 | 0 | 0 | 0 | 0 | 0
SolarWorker_4 | 24 | 0 | 0 | 237 | 0 | 0
SolarWorker_5 | 25 | 0 | 121 | 121 | 0 | 0

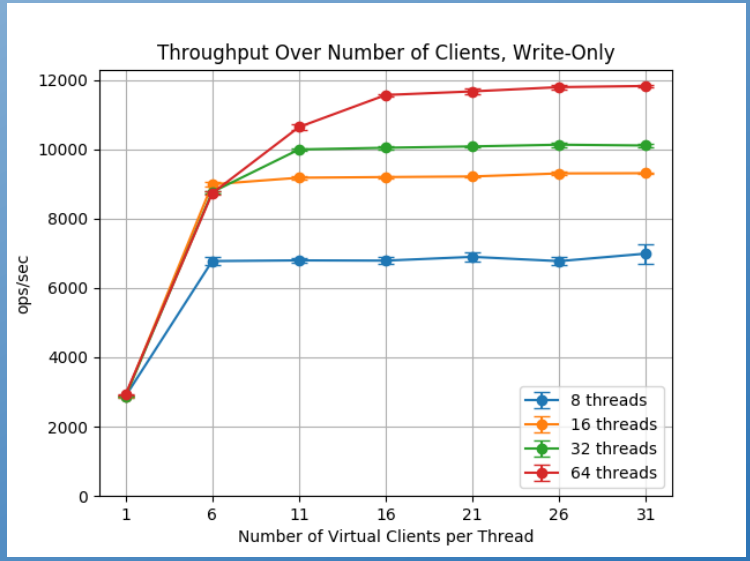
Performance Analysis in IDEs

- Environments typically include one
  - VS Code, IntelliJ, (Eclipse TPTP)
- Code Bubbles
  - Automatic as part of debugging
    - Collects more information than displayed
- Again, these provide limited data
  - Especially for multithreaded cases

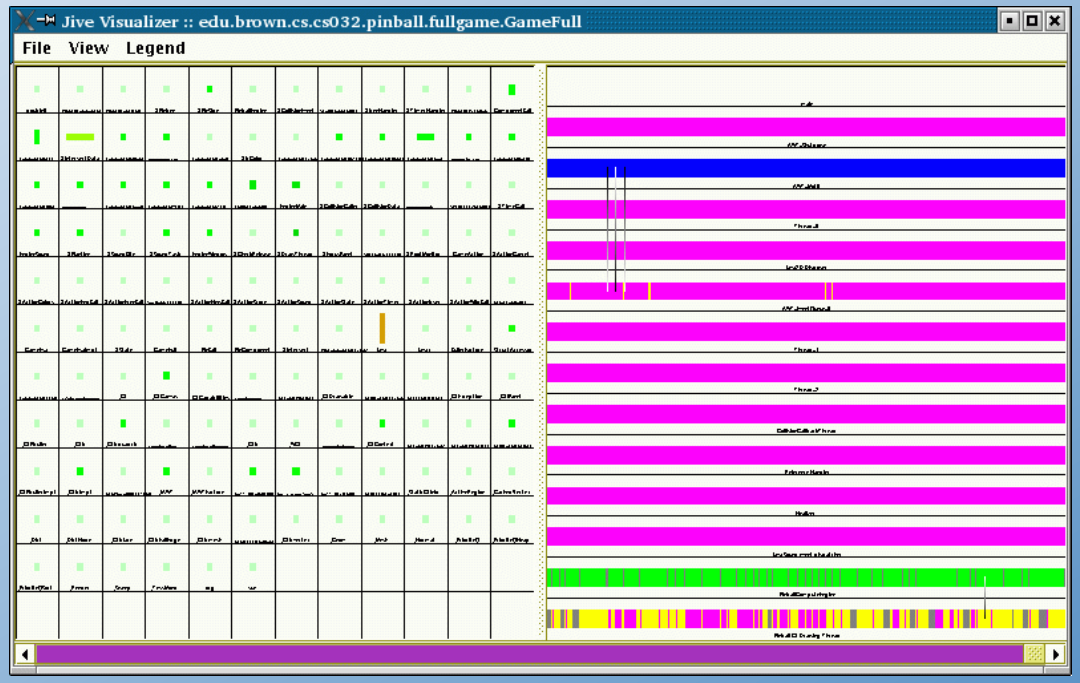


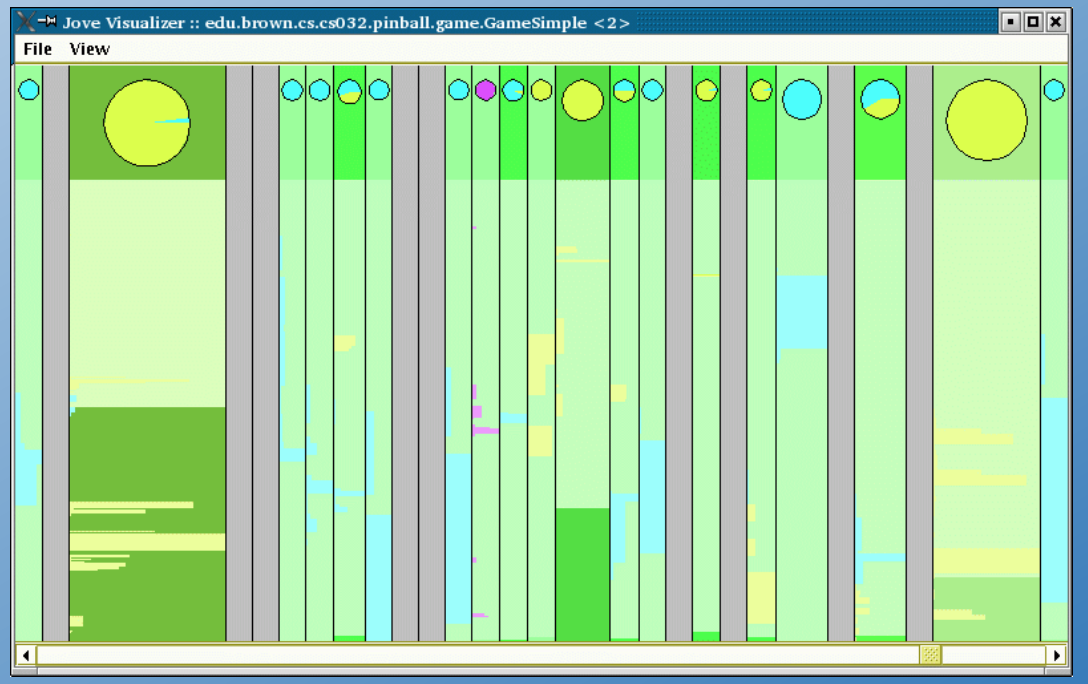
Multithreaded Performance Analysis

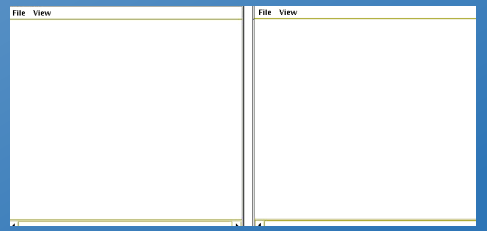
- Detecting the state of all the threads
  - Running, busy, waiting, I/O
- Detecting what threads are blocking on
  - Shared locks, bottlenecks, other threads
- Detecting what threads are doing
  - In terms of the program
  - Why they are blocked, waiting, or doing I/O
- Problem: multithreaded web crawler would work well for a while, but then it would essentially halt for a time before continuing



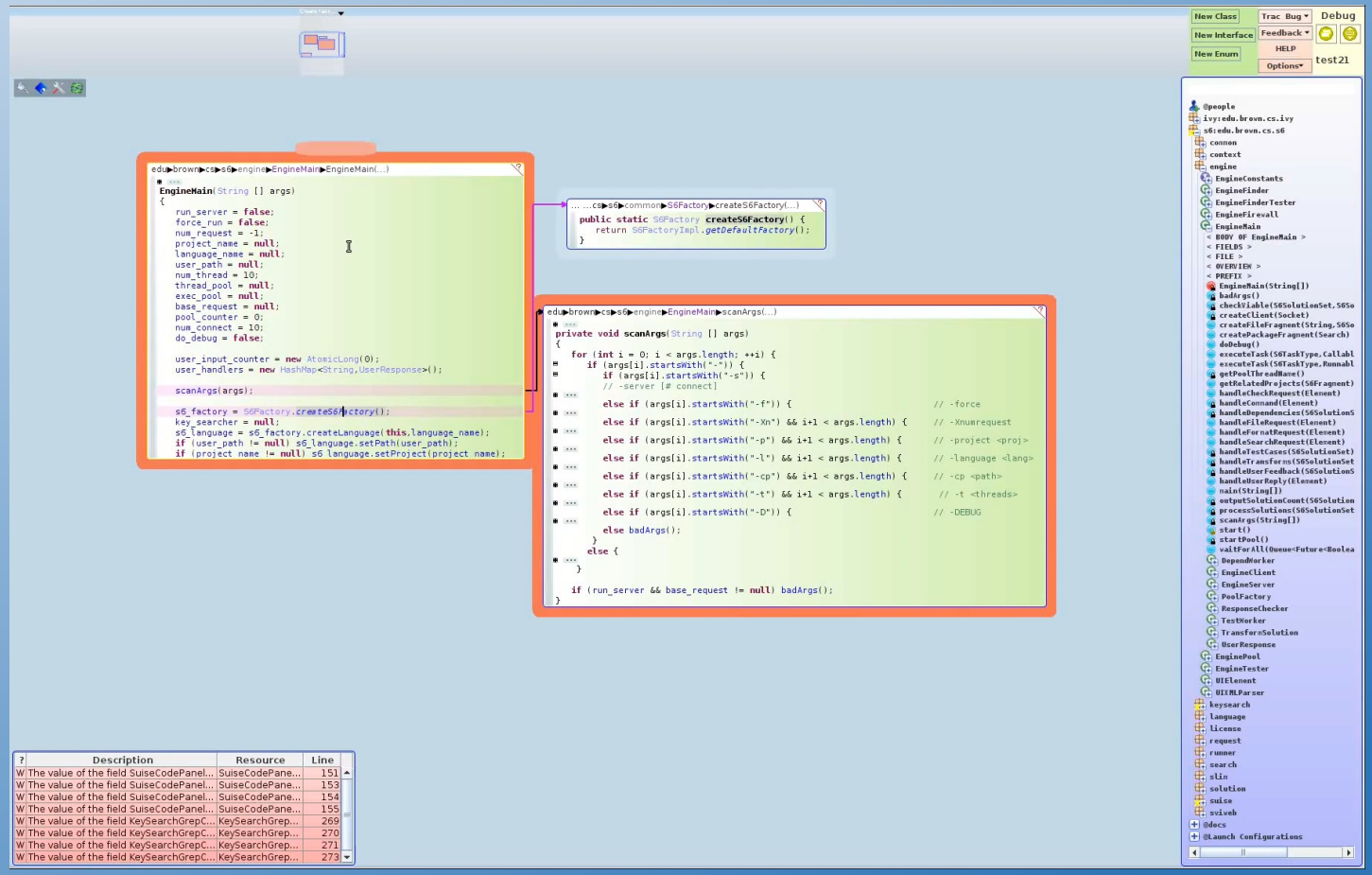
Jive, Jove and Veld







Dynamic Visualization in Code Bubbles



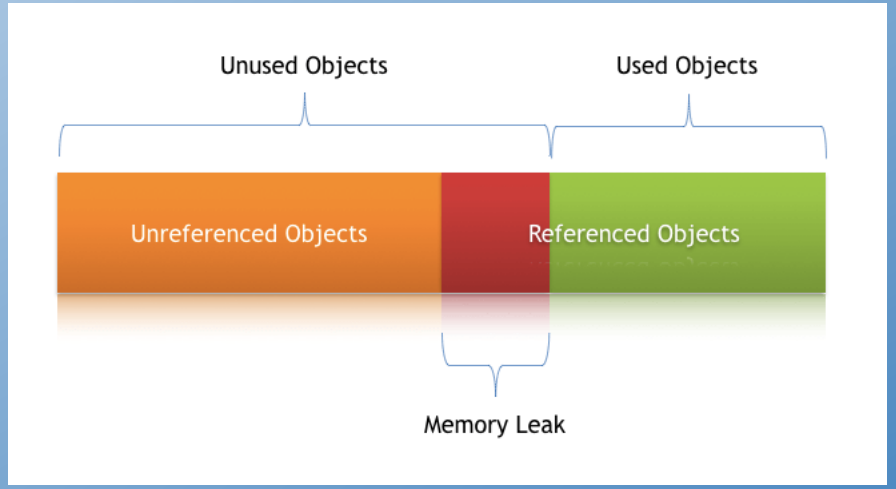
Code Reviews

- A form of manual testing
- Take a piece of code (method, class, …)
  - Have a panel of reviewers
  - Pass the code out to the panel (possibly in advance)
  - Panel goes over the code line-by-line
- Goal is to find and eliminate all potential bugs
  - And make sure the code conforms to various guidelines

EXERCISE

- Example code review (codereview.java)

Memory Problems

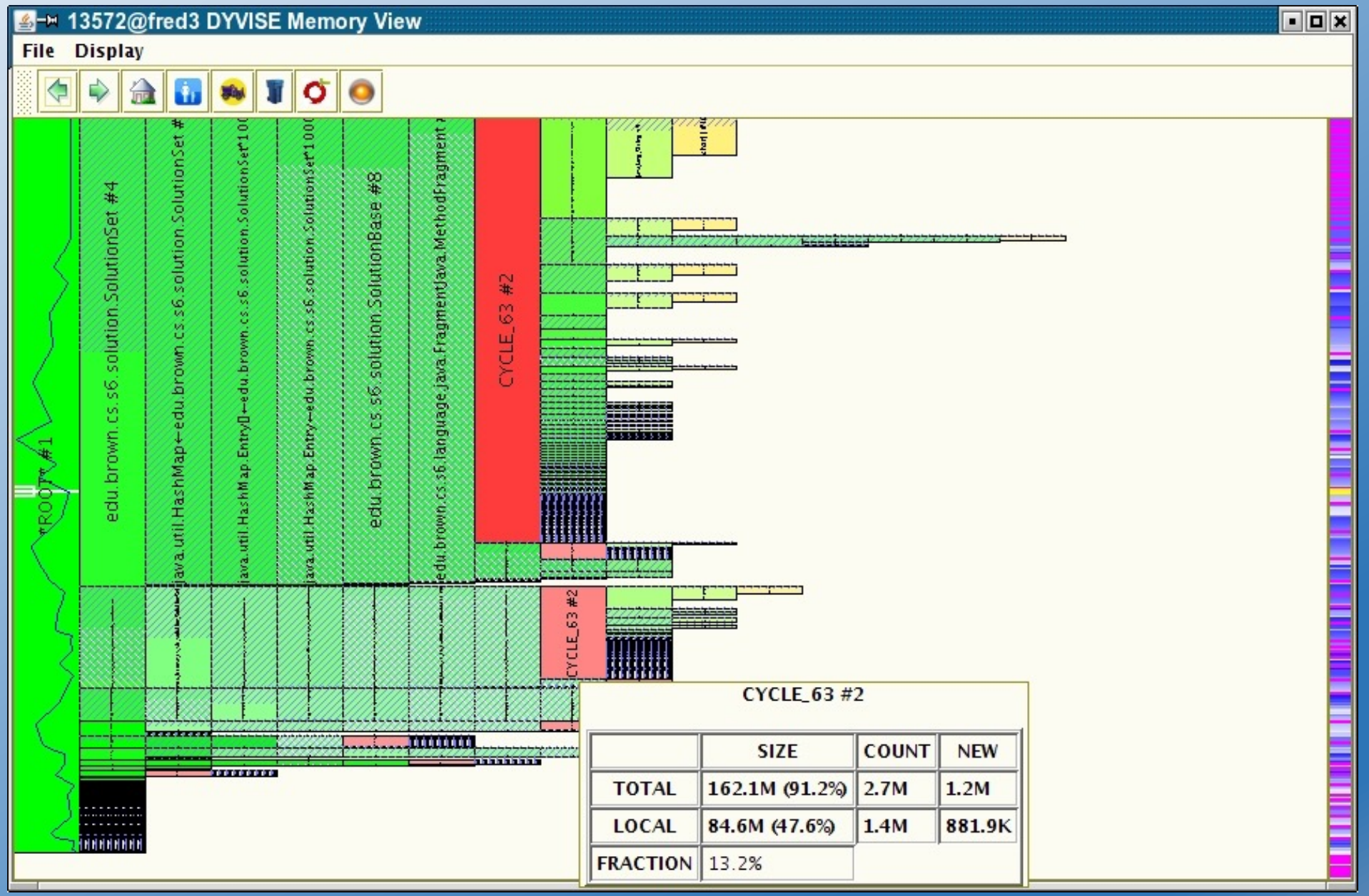


- Memory Leaks occur in Java
  - Despite garbage collection
  - But are more subtle than in C/C++
- Tools for detecting leaks
  - jmap –dump; jhat are standard java tools
  - Eclipse memory analysis extensions
    - Summary
    - Browse through memory
  - Dyvise memory data
- Test cases for memory problems
  - Especially for recreating them

Memory Ownership

- Who is in charge of a particular piece of memory (object)
  - Creating it, holding a pointer to it, freeing it
  - Making this clear reduces memory problems
    - Especially in C/C++ but also in Java and other systems
    - In addition to garbage collection
- Ideally this should be clear
  - Non-owners shouldn't **save** pointers (or should use weak ones in Java)
  - Generally handled via documentation & manual techniques
  - Requires strong programmer discipline to get right
    - Especially with multithreading
  - RUST makes it explicit

DYMEM


13572@fred3 DYVISE Memory View
File
Display
ROOT #1
edu.brown.cs.s6.solution.SolutionSet #4
java.util.HashMap←edu.brown.cs.s6.solution.SolutionSet #
java.util.HashMap.Entry[]←edu.brown.cs.s6.solution.SolutionSet*10
java.util.HashMap.Entry←edu.brown.cs.s6.solution.SolutionSet*100
edu.brown.cs.s6.solution.SolutionBase #8
edu.brown.cs.s6.language.java.FragmentJava.MethodFragment
CYCLE_63 #2
CYCLE_63 #2
CYCLE_63 #2
SIZE
COUNT
NEW
TOTAL
162.1M (91.2%)
2.7M
1.2M
LOCAL
84.6M (47.6%)
1.4M
881.9K
FRACTION
13.2%

PROJECT

- Ensure you have a testing strategy for your project
  - Test environment
  - Test version
  - Include documentation of the strategy in your repository
- Do a code review within your project team
  - Choose a complex module within the project
  - Point everyone to the code in advance
  - Then go over the code line by line to ensure it works
  - Submit a summary of your findings in Canvas
- Does any group want to present on 11/26?
  - Tell me by next Tuesday