
# Verification

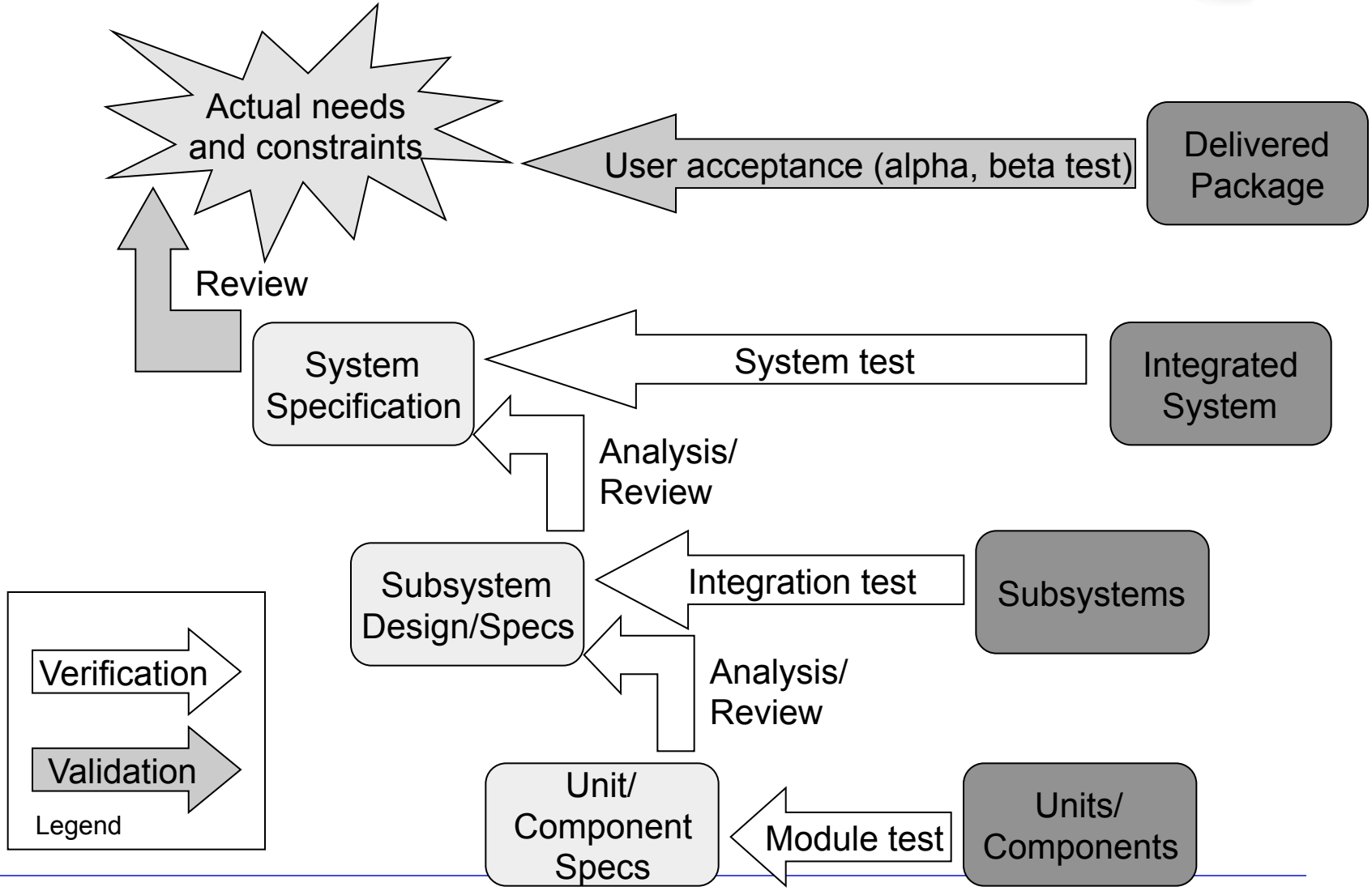
(Note: verification used in a very general sense, including validation)

# Definitions

- ***Validation.***
  - *"The assurance that a product, service, or system meets the needs of the customer and other identified stakeholders. It often involves acceptance and suitability with external customers. Contrast with verification"* (IEEE)
  - "Are you building the right thing?"
- ***Verification.***
  - *"The evaluation of whether or not a product, service, or system complies with a regulation, requirement, specification, or imposed condition. It is often an internal process. Contrast with validation." (IEEE)*
  - "Are you building it right?"

"A Guide to the Project Management Body of Knowledge (PMBOK® Guide) - Fourth Edition" IEEE

# V&V activities and software artifacts (the V model)

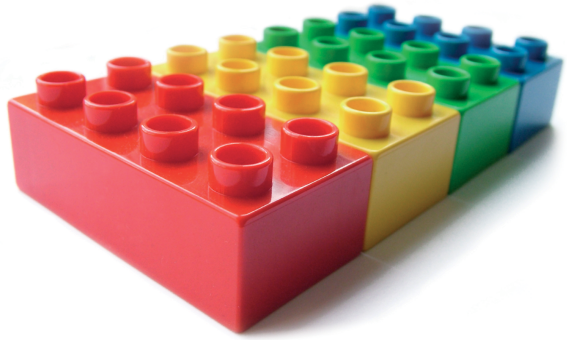


# V-Model Pros & Cons


- Pros
  - Easy to understand
  - Easy to manage
  - Every stage is tested
  - Good for small/medium projects with well defined requirements
- Cons
  - Not flexible
  - No early prototypes
  - Midway changes requires documents update
  - Not good for big projects with unclear or unstable requirements

# Unit Testing

- Conducted by the same developers (white box)
- Aimed at testing sections of code (functional or class level)
- Dependencies are mocked up
- Ensures that dependencies work independently from each other
- Why unit testing?
  - Find problems early
  - Facilitates changes
  - Guides the design
  - Visual feedback



# Integration Testing

- Aimed at exercising
  - interfaces
  - modules interactions
- How
  - Incrementally (facilitates bug tracking)
  - Big bang (faster execution)

# System Testing

- Conducted on a complete integrated system
- Independent teams (black box)
- Functional and non-functional requirements
- Testing environment should be as close as possible to production environment

# Performance Testing


- Purpose
  - Eliminate bottlenecks affecting response time, utilization, throughput (whitebox)
  - Benchmarking, *i.e.*, establish a performance baseline and possibly compare it with different versions of the same product or a different competitive product (blackbox)
- Prerequisites
  - Expected workload
  - Acceptable performance
- Tuning
  - Inefficient algorithms
  - Query optimizations
  - Hardware/Network

# Load Testing


- Purpose
  - Expose bugs such as memory leaks, mismanagement of memory, buffer overflows
  - Identify upper limits of components
- How
  - Increase the load until threshold
  - Load the system with the maximum load it can operate for a long period

# Stress Testing

**Purpose**

- Make sure that the system recovers gracefully after failure

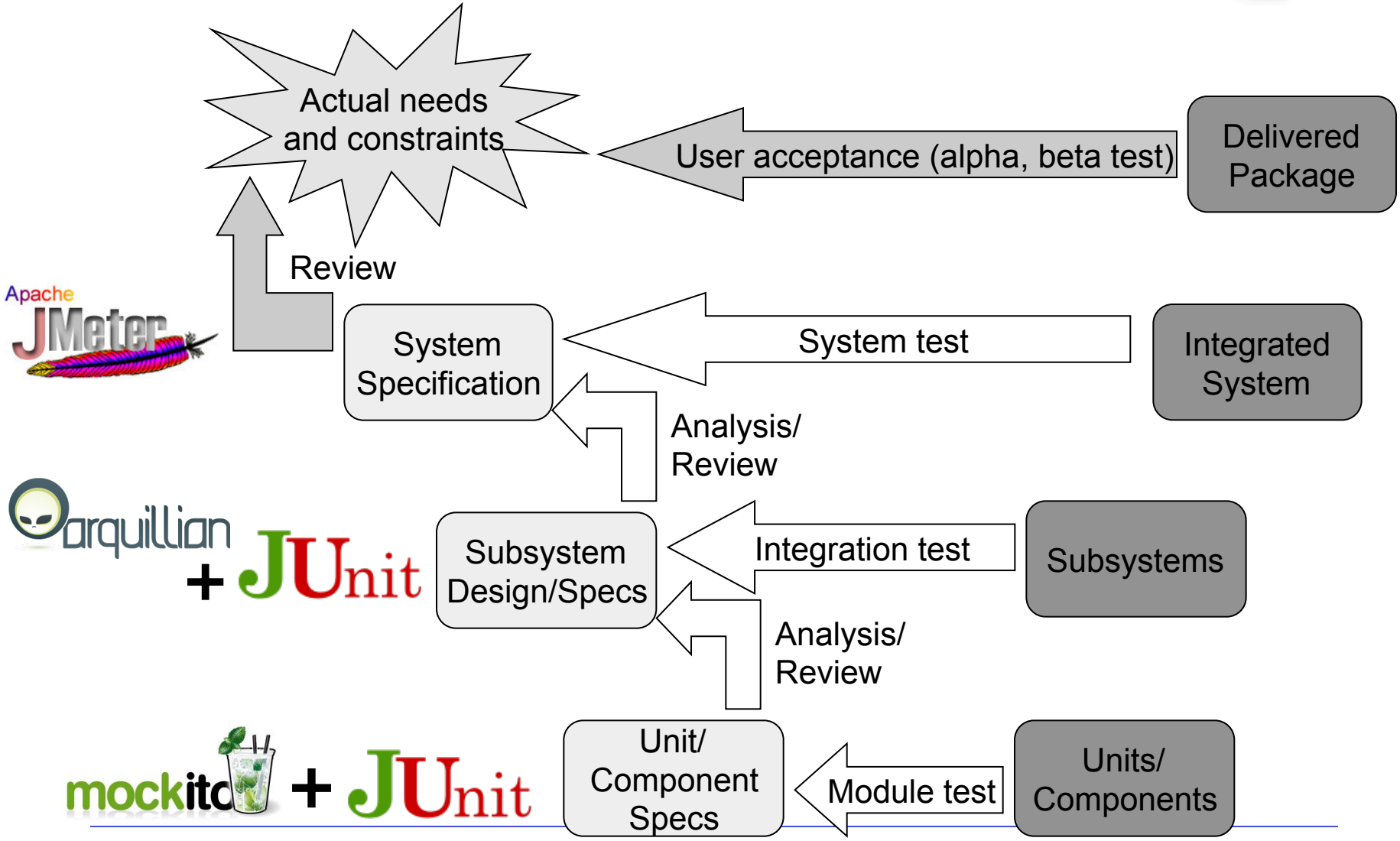
**How**

- Trying to break the system under test by overwhelming its resources or by taking resources away from it

**Examples**

- Double the baseline number for concurrent users/HTTP connections
- Randomly shut down and restart ports on the network switches/routers that connects servers

# V&V activities and software artifacts (the V model)



Politecnico
di Milano


# Mockito: creating mockups for unit testing


Website: http://mockito.github.io
Doc: http://mockito.github.io/mockito/docs/current/org/mockito/Mockito.html

# Why Mocking?


- Unit tests should
  - cover the smaller testable functionality
  - be fast (no environments or DB setup)
  - isolate dependencies for fast bug tracking
- Mocking allows you to
  - Abstract dependencies and have predictable results
  - Check that the interaction between the testee and the mock is correct
- What can you mock?
  - A persistence manager
  - An external service
  - A not yet implemented class
  - Someone else's code
  - …

# State vs Interaction testing


- State testing asserts properties on an object
  - *assertEquals(4, item.getCount())*;
- Interaction testing verifies the interactions between objects
  - Did my controller correctly call my service?
- Mockito provides a framework for interaction testing

# Stubbing


- Adding predefined response to Mock object methods

- Examples
  - *when(list.get(0)).thenReturn("Hello");*
  - *when(mockedMap.get("key")).thenReturn("someValue");*
  - *doThrow(new IllegalStateException("Illegal").when(obj).get(2);*

# Verify


- Helps verify that certain methods were called “n” times or where never called

- Examples
  - *Default is 1*
  - *times(n)*

# Mockito – Common 3A Test


```
@Test
public void test() {
    //Arrange
    UnitToTest cut = new UnitToTest();
    MockedClass aMock = mock(MockedClass);
    when(aMock.isCalled()).thenReturn(true);
    doThrow(new RuntimeException()).when(aMock).shouldNotBeCalled();

    //Act
    cut.doSomething(aMock);

    //Assert
    verify(aMock, times(1)).mockMethod();
}
```

Politecnico
di Milano


# Arquillian: an integration testing framework for containers


http://arquillian.org

# Arquillian


- Used to execute test cases against the container
  - Testing a component in business app is challenging
  - Interaction with the system as important as the performed work

- E.g. dependency injection and transaction control
  - Is the right component injected?
  - Is the interaction with the database ok?

Politecnico
di Milano

# Jmeter: a load testing tool for analyzing and measuring performance


http://jmeter.apache.org

**Credits:**
**Anthony Colebourne**

# What is JMeter


- A GUI desktop application designed to load test functional behavior and measure performance. It was originally designed for testing Web Applications but has since expanded to other test functions
  - Has a rich graphical interface
  - Built in Java
- It can be used to simulate a heavy load on a server, network or object to test its strength or to analyze overall performance under different load types

# Features of JMeter


- Graphical Analysis / Exporting Test Results
- Remote Distributed Execution
  - If you want to generate load using multiple test servers. You can run multiple server components of JMeter remotely. And you can control it by a single JMeter GUI to gather results.
    - http://jmeter.apache.org/usermanual/remote-test.html
- Highly Extensible
  - Custom Additions (Write your own samplers / listeners)
  - PlugIns

# What Can You Do With It?

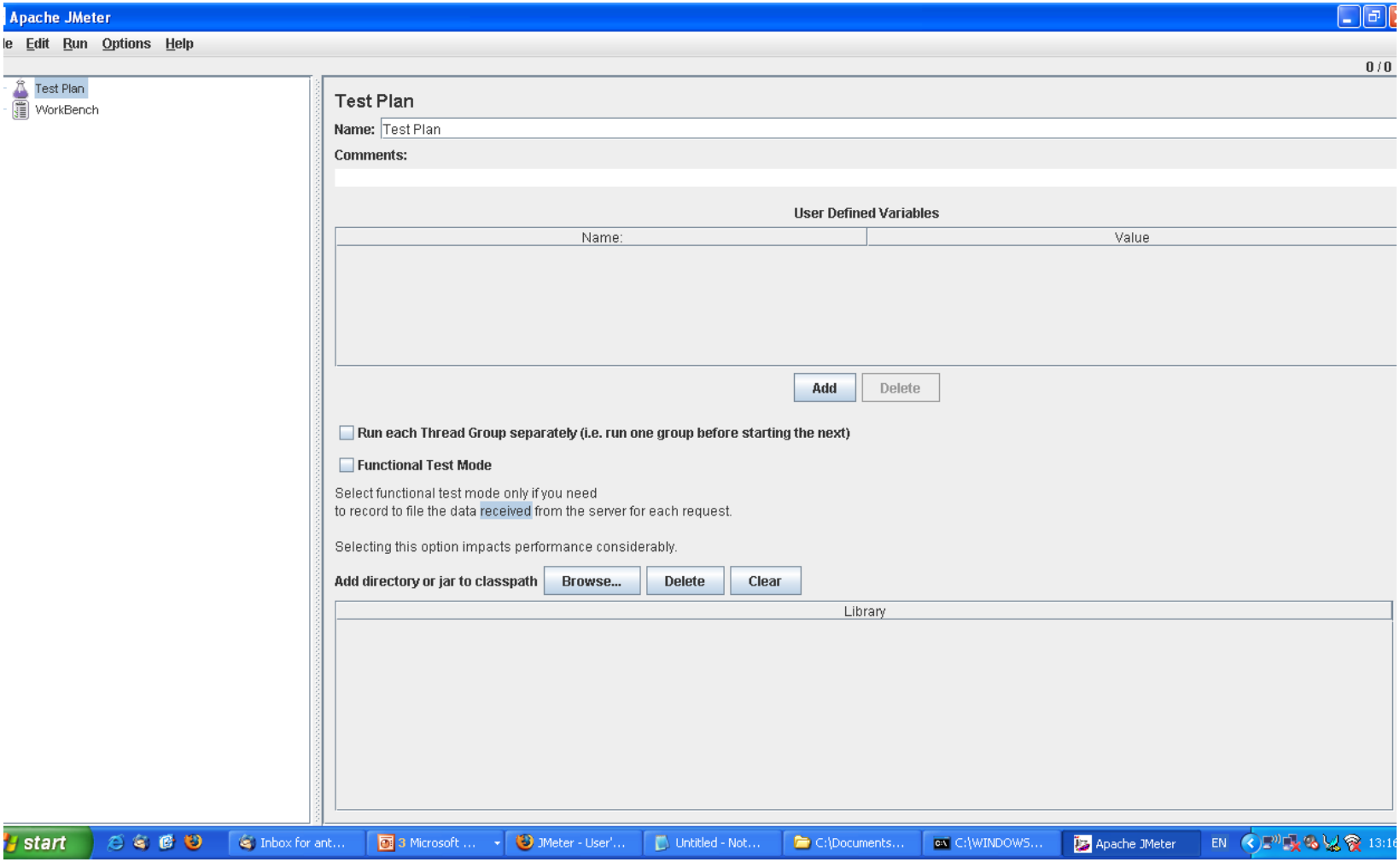

- JMeter lets you set up test plans that simulate logging into a web site, filling out forms, clicking buttons, links, etc.

- You can simulate the number of users doing this, the rate that they do it…

# Setting Up and Running JMeter


- Download the binary from website
  - http://jmeter.apache.org/download_jmeter.cgi
  - It's platform independent, so the same download will run on Windows, Linux, Mac.

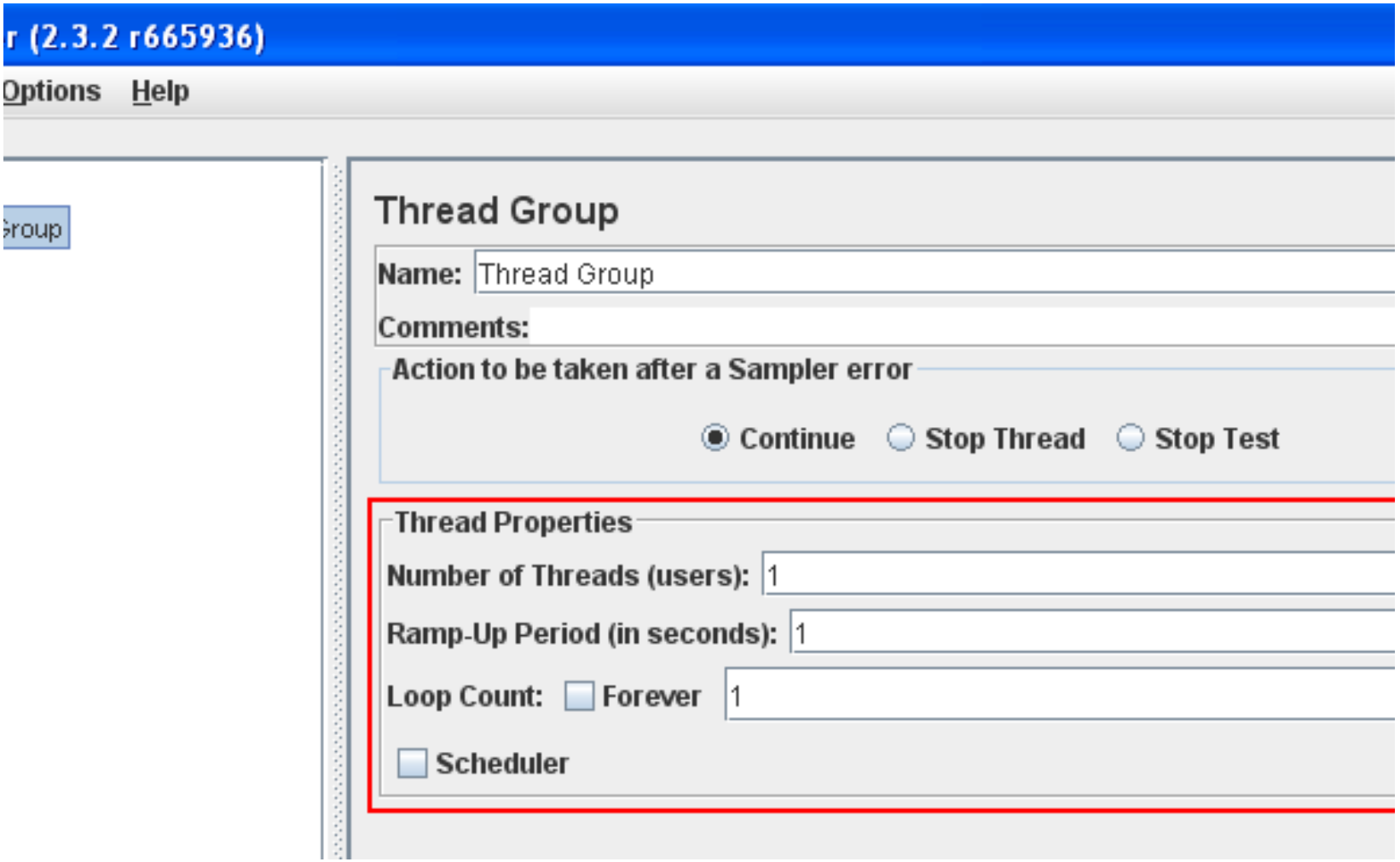
# JMeter Control Panel



# Terminologies used in J-Meter

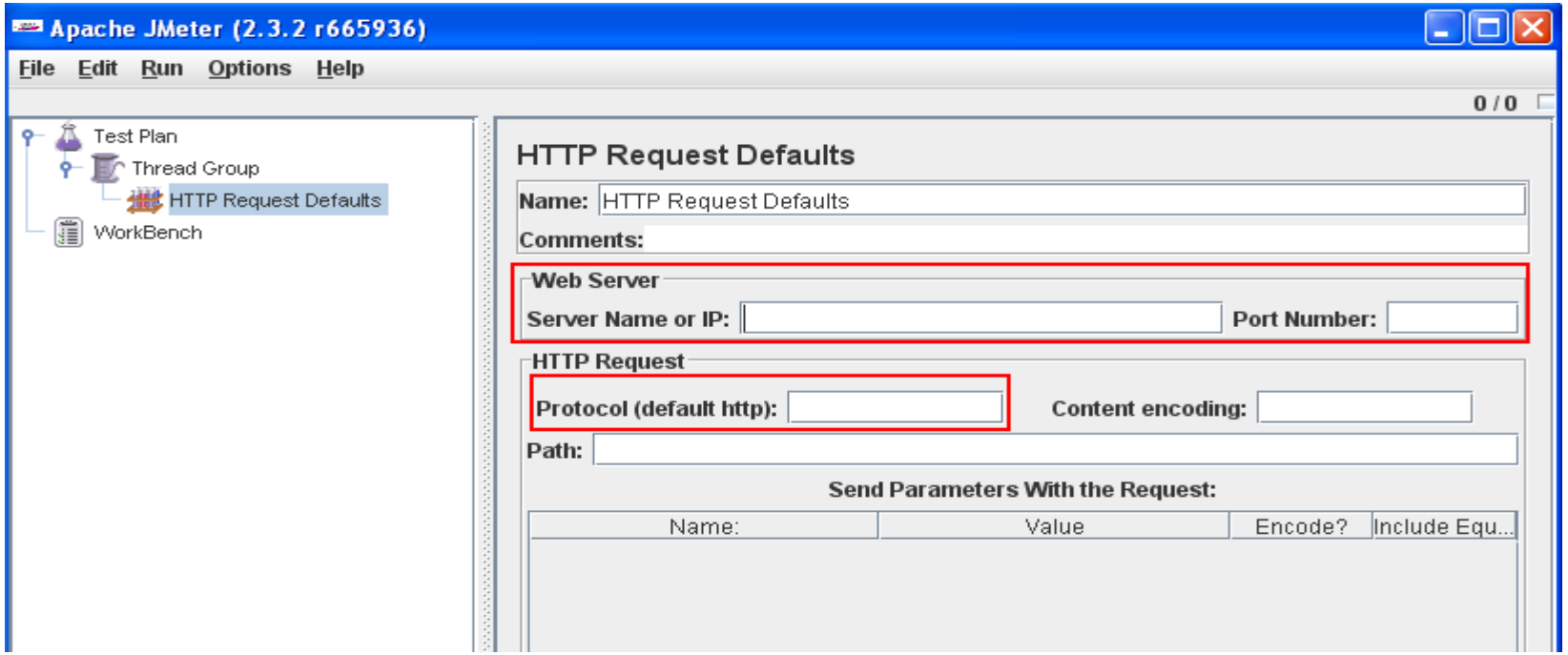

- Number of Threads: Number of virtual users
- Thread Group: This is the place where number of threads, ramp-up period, loop count are given while executing
- Ramp-Up Period: It indicates the time taken by J-Meter to create all of the threads needed. If we set 10 seconds as the ramp-up period for 5 threads then the J-Meter will take 10 seconds to create those 5 threads. Also by setting its value to 0, all the threads can be created at once
- Loop Count: By specifying its value J-meter gets to know that how many times a test is to be repeated

# Terminologies used in J-Meter



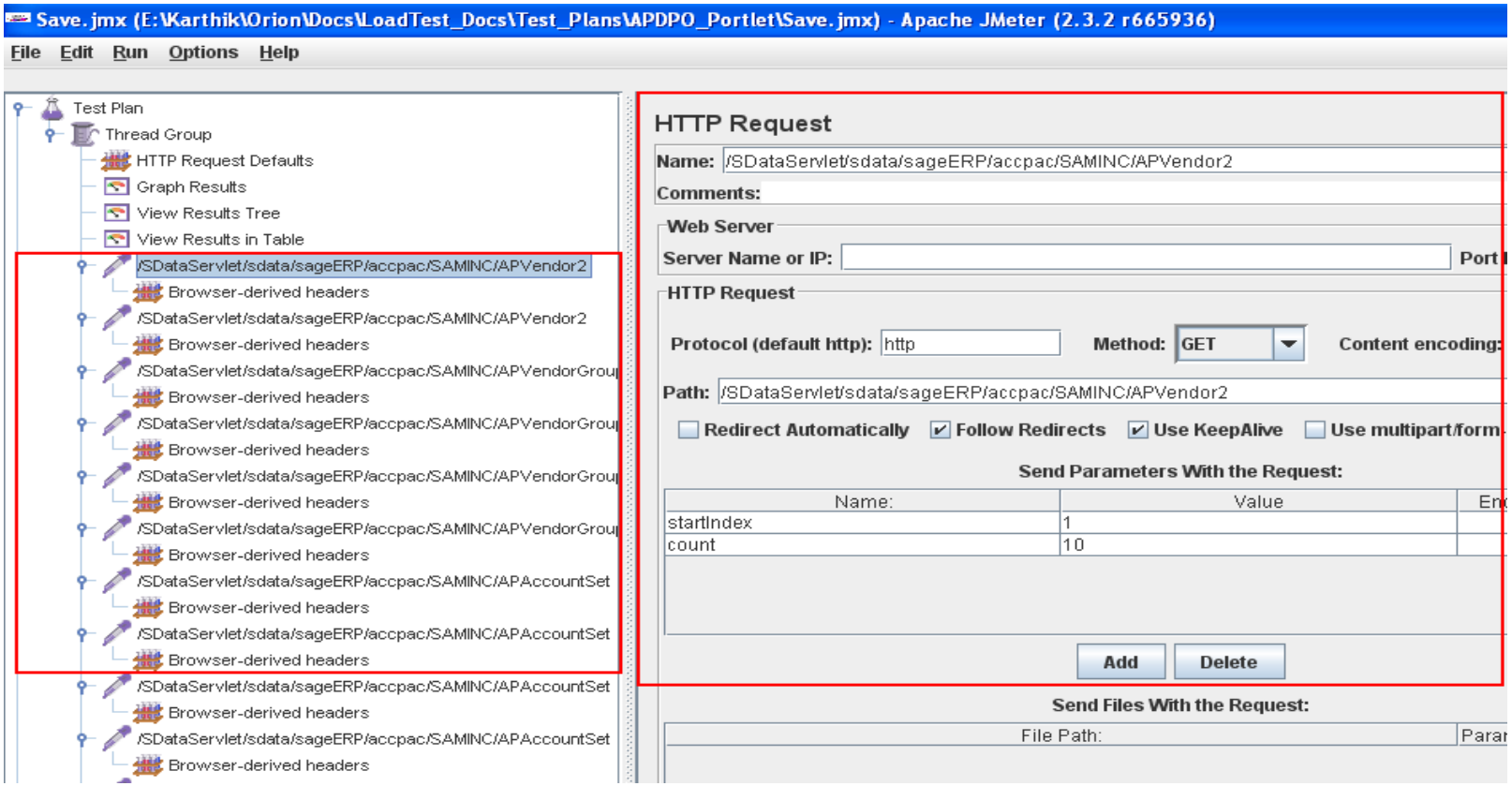
# Terminologies used in J-Meter

- Http Request Defaults: this is the place where machine IP number, machine port number and name of the protocol (HTTP) are saved



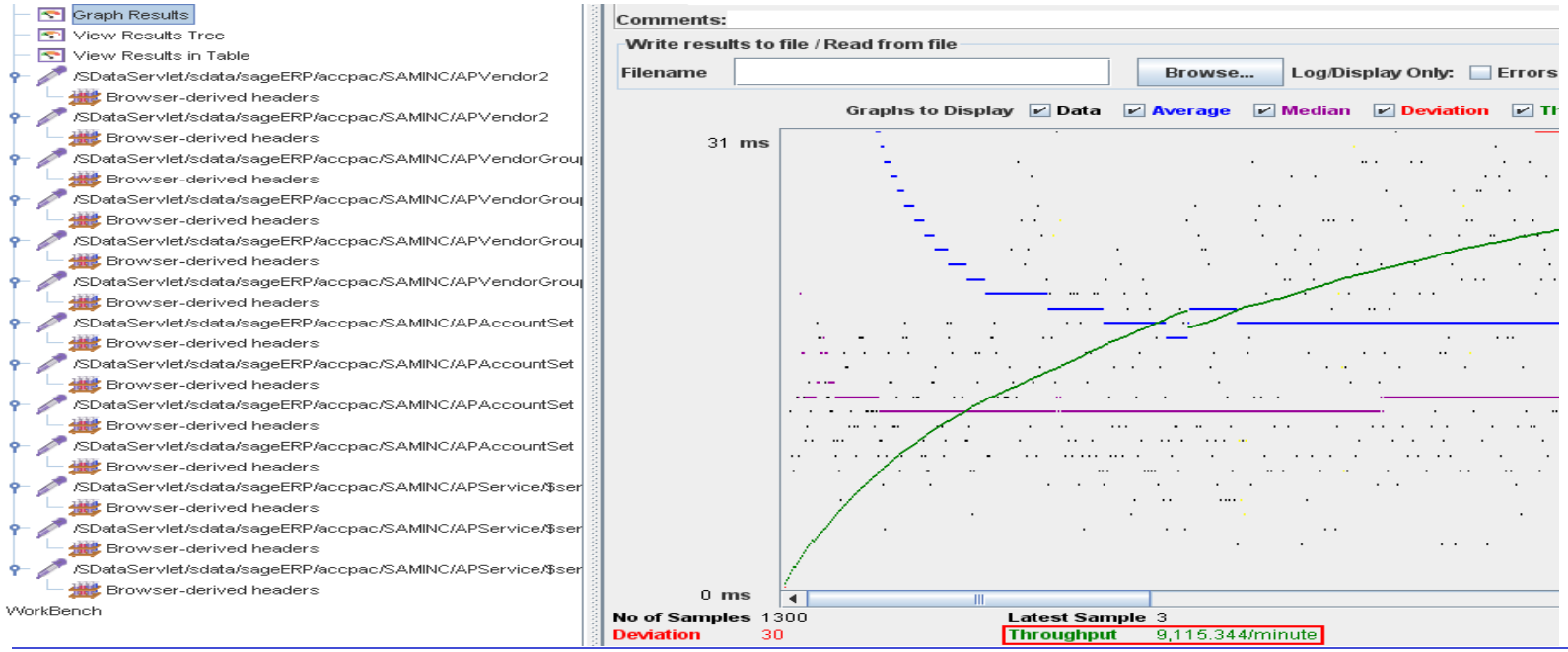
# Terminologies used in J-Meter

- Sampler: This is the place where all requests are saved



# Terminologies used in J-Meter

- Listeners: this is the place where result will be shown
- Graph Results Listener: using this listener, we can get throughput value. Throughput got from this listener normally will be in requests/minute

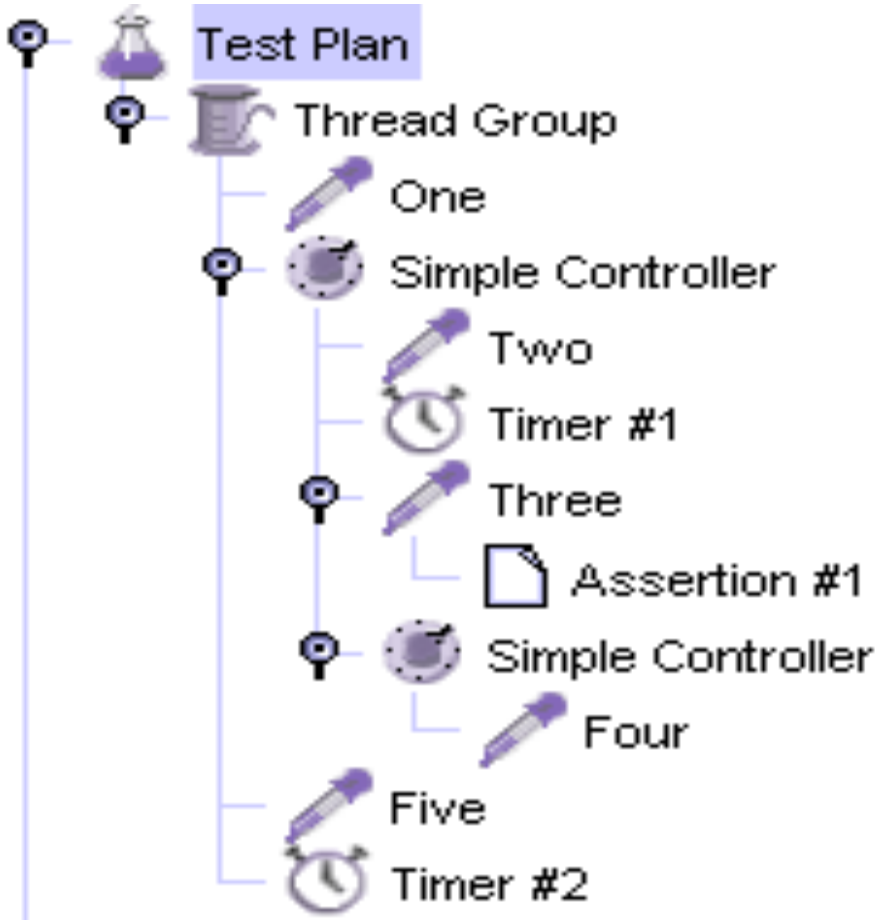


# Test Plan


- A complete test plan will consist of one or more Thread Groups, logic controllers, samplers, listeners, timers, assertions, and configuration elements

- Test Plans are represented as an Ordered Tree

# Test Plan


Test Plan
Thread Group
One
Simple Controller
Two
Timer #1
Three
Assertion #1
Simple Controller
Four
Five
Timer #2

# Scoping Rules


- Some elements are primarily ordered (e.g., controllers and samplers)

- Other elements are hierarchical. An Assertion, for example, is hierarchical in the test tree. If its parent is a request, then it is applied to that request. If its parent is a Controller, then it affects all requests that are descendants of that Controller

# Terminologies used in J-Meter


- Logic Controllers: Let you customize the logic that JMeter uses to decide when to send requests. Logic Controllers can change the order of requests coming from their child elements. They can modify the requests themselves, cause JMeter to repeat requests, etc.

- Test Plan
- Thread Group
  - Once Only Controller
    - Login Request (an HTTP Request )
  - Load Search Page (HTTP Sampler)
  - Interleave Controller
    - Search "A" (HTTP Sampler)
    - Search "B" (HTTP Sampler)
    - HTTP default request (Configuration Element)
  - HTTP default request (Configuration Element)
  - Cookie Manager (Configuration Element)

# Elements of a Test Plan

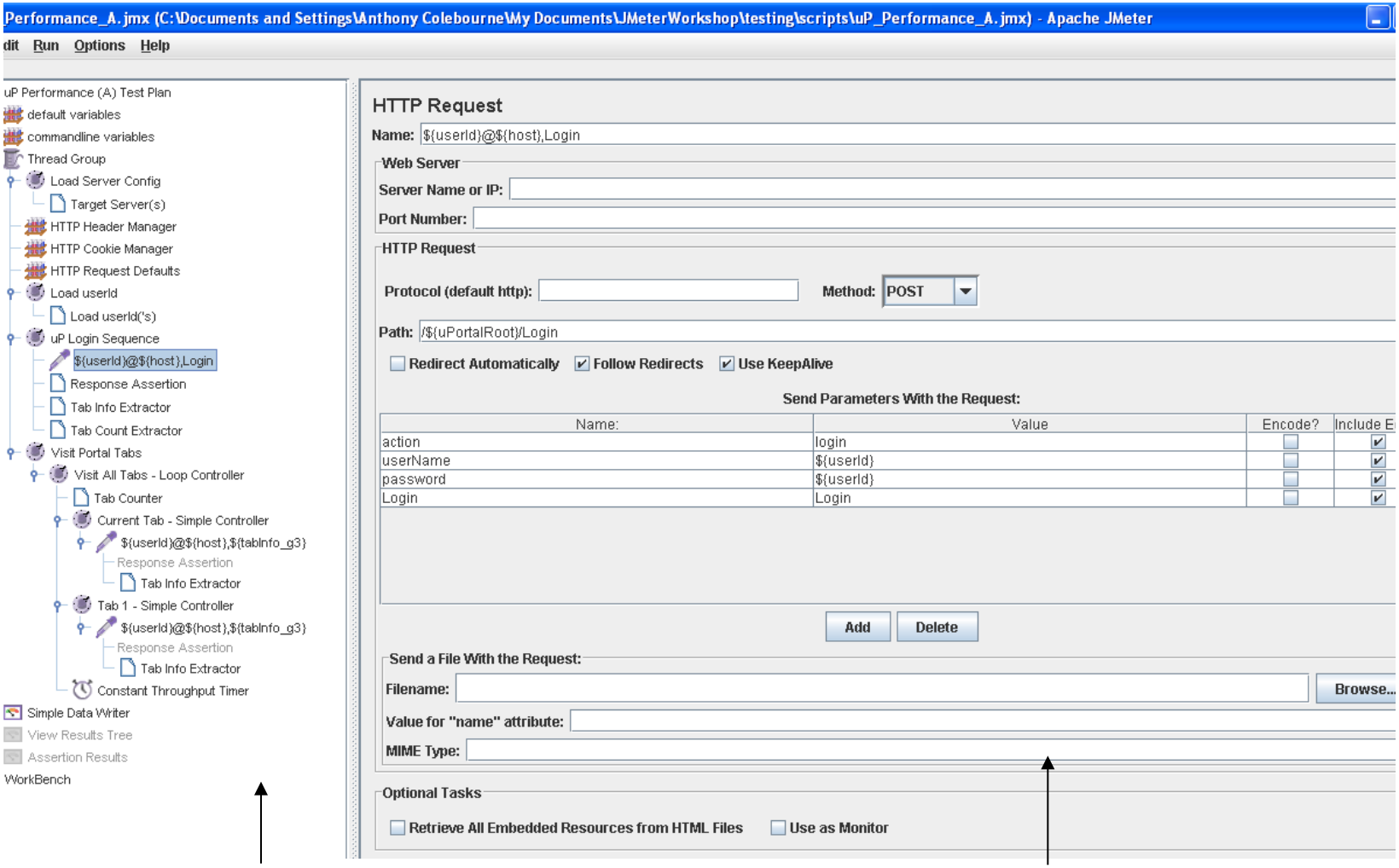

- ThreadGroup
  - Set the number of threads
  - Set the ramp-up period
  - Set the number of times to execute the test
- Samplers
  - HTTP Request
  - JDBC Request
  - LDAP Request
  - WebService (SOAP) Request
- Logic Controllers
  - Simple Controller - The Simple Logic Controller lets you organize your Samplers and other Logic Controllers
  - Loop Controller
  - Once Only Controller
  - Interleave Controller
  - Throughput Controller
  - Others … (e.g. Transaction Controller)
- Listeners
  - Simple Data Writer
  - Graph Results

http://jmeter.apache.org/usermanual/component_reference.html

# Elements of a Test Plan


- Timers
  - The timer will cause a delay between each request that a thread makes
- Assertions
  - The ‘test’ of the response from the server
- Configuration Elements
  - Sets default
- Pre-Processor / Post-Processor Elements

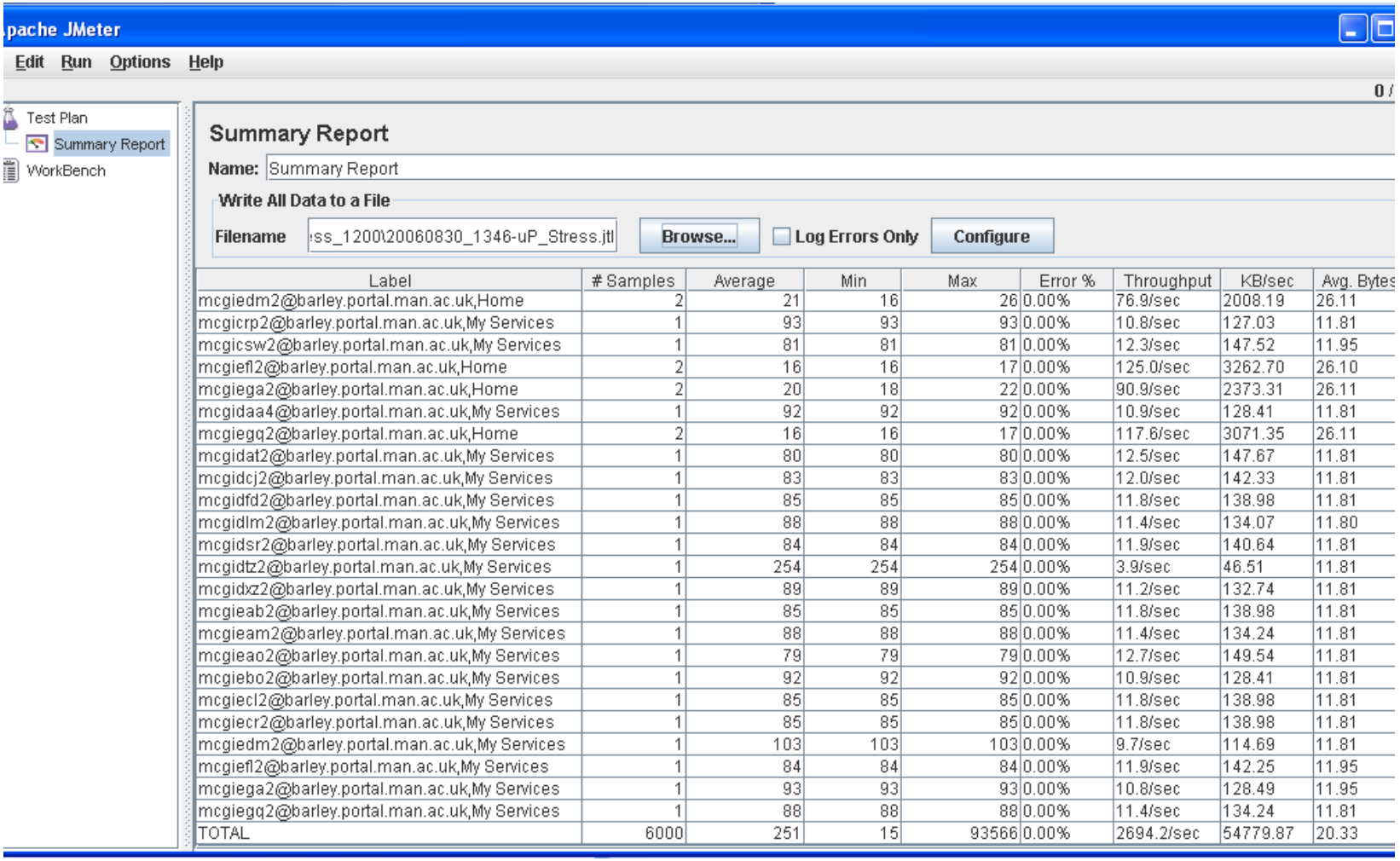
# Login Sequence




**The Test Plan**

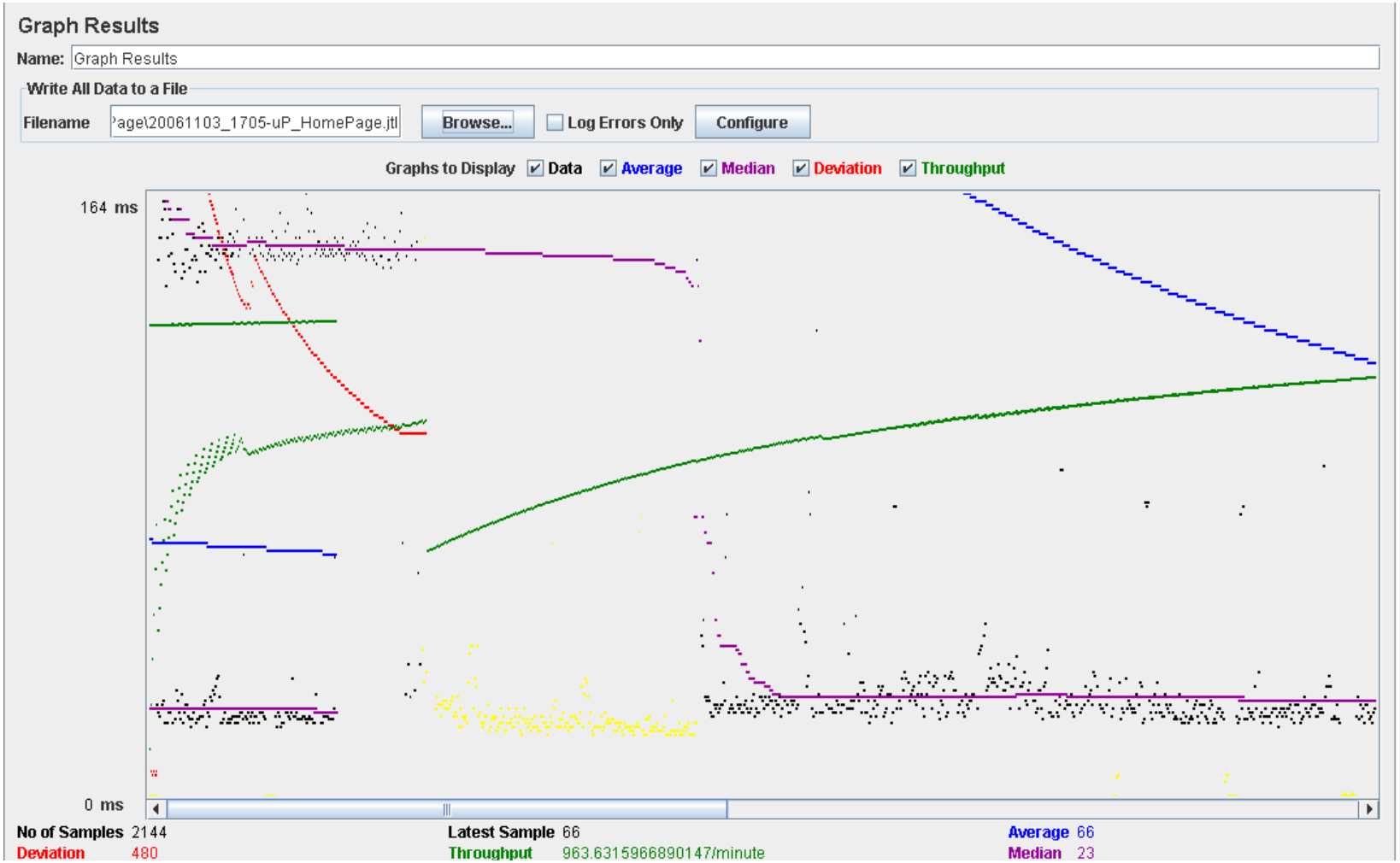
**Login Sequence**

# View results summary



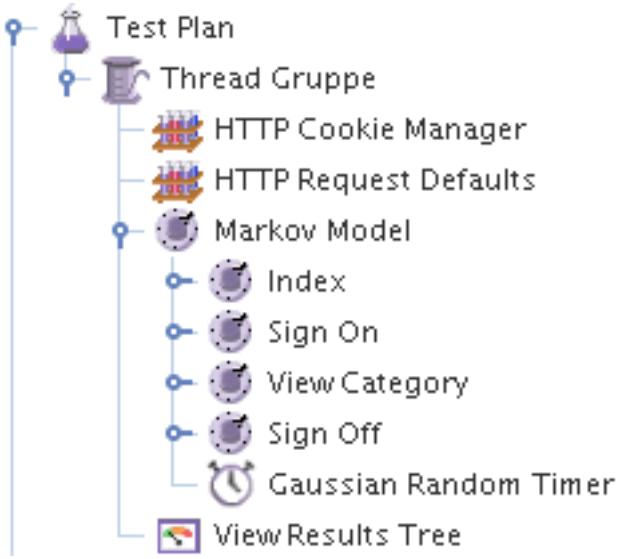

| Label | # Samples | Average | Min | Max | Error % | Throughput | KB/sec | Avg. Bytes |
|---|---|---|---|---|---|---|---|---|
| mcgiedm2@barley.portal.man.ac.uk,Home | 2 | 21 | 16 | 26 | 0.00% | 76.9/sec | 2008.19 | 26.11 |
| mcgicrp2@barley.portal.man.ac.uk,My Services | 1 | 93 | 93 | 93 | 0.00% | 10.8/sec | 127.03 | 11.81 |
| mcgicsw2@barley.portal.man.ac.uk,My Services | 1 | 81 | 81 | 81 | 0.00% | 12.3/sec | 147.52 | 11.95 |
| mcgiefl2@barley.portal.man.ac.uk,Home | 2 | 16 | 16 | 17 | 0.00% | 125.0/sec | 3262.70 | 26.10 |
| mcgiega2@barley.portal.man.ac.uk,Home | 2 | 20 | 18 | 22 | 0.00% | 90.9/sec | 2373.31 | 26.11 |
| mcgidaa4@barley.portal.man.ac.uk,My Services | 1 | 92 | 92 | 92 | 0.00% | 10.9/sec | 128.41 | 11.81 |
| mcgiegq2@barley.portal.man.ac.uk,Home | 2 | 16 | 16 | 17 | 0.00% | 117.6/sec | 3071.35 | 26.11 |
| mcgidat2@barley.portal.man.ac.uk,My Services | 1 | 80 | 80 | 80 | 0.00% | 12.5/sec | 147.67 | 11.81 |
| mcgidcj2@barley.portal.man.ac.uk,My Services | 1 | 83 | 83 | 83 | 0.00% | 12.0/sec | 142.33 | 11.81 |
| mcgidfd2@barley.portal.man.ac.uk,My Services | 1 | 85 | 85 | 85 | 0.00% | 11.8/sec | 138.98 | 11.81 |
| mcgidlm2@barley.portal.man.ac.uk,My Services | 1 | 88 | 88 | 88 | 0.00% | 11.4/sec | 134.07 | 11.80 |
| mcgidsr2@barley.portal.man.ac.uk,My Services | 1 | 84 | 84 | 84 | 0.00% | 11.9/sec | 140.64 | 11.81 |
| mcgidtz2@barley.portal.man.ac.uk,My Services | 1 | 254 | 254 | 254 | 0.00% | 3.9/sec | 46.51 | 11.81 |
| mcgidxz2@barley.portal.man.ac.uk,My Services | 1 | 89 | 89 | 89 | 0.00% | 11.2/sec | 132.74 | 11.81 |
| mcgieab2@barley.portal.man.ac.uk,My Services | 1 | 85 | 85 | 85 | 0.00% | 11.8/sec | 138.98 | 11.81 |
| mcgieam2@barley.portal.man.ac.uk,My Services | 1 | 88 | 88 | 88 | 0.00% | 11.4/sec | 134.24 | 11.81 |
| mcgieao2@barley.portal.man.ac.uk,My Services | 1 | 79 | 79 | 79 | 0.00% | 12.7/sec | 149.54 | 11.81 |
| mcgiebo2@barley.portal.man.ac.uk,My Services | 1 | 92 | 92 | 92 | 0.00% | 10.9/sec | 128.41 | 11.81 |
| mcgiecl2@barley.portal.man.ac.uk,My Services | 1 | 85 | 85 | 85 | 0.00% | 11.8/sec | 138.98 | 11.81 |
| mcgiecr2@barley.portal.man.ac.uk,My Services | 1 | 85 | 85 | 85 | 0.00% | 11.8/sec | 138.98 | 11.81 |
| mcgiedm2@barley.portal.man.ac.uk,My Services | 1 | 103 | 103 | 103 | 0.00% | 9.7/sec | 114.69 | 11.81 |
| mcgiefl2@barley.portal.man.ac.uk,My Services | 1 | 84 | 84 | 84 | 0.00% | 11.9/sec | 142.25 | 11.95 |
| mcgiega2@barley.portal.man.ac.uk,My Services | 1 | 93 | 93 | 93 | 0.00% | 10.8/sec | 128.49 | 11.95 |
| mcgiegq2@barley.portal.man.ac.uk,My Services | 1 | 88 | 88 | 88 | 0.00% | 11.4/sec | 134.24 | 11.81 |
| TOTAL | 6000 | 251 | 15 | 93566 | 0.00% | 2694.2/sec | 54779.87 | 20.33 |

# View results as graph



# Plugins

- Many useful plugins are continuously developed
- Markov4JMeter: Probabilistic and Intensity-Varying Workload Generation for Session-Based Systems



# Tips


- Use timers to avoid hammering the server
- Limit the Number of Threads
  - Hardware will limit the number of threads you can effectively run. A faster machine makes JMeter work harder since it returns request quicker
- User variables
  - Create a text file containing the user names and passwords.
  - Add a CSV DataSet configuration element. Name the variables USER and PASS. Use ${USER} and ${PASS} within samplers
- Reducing resource requirements
  - Use non-GUI mode
  - Use as few Listeners as possible
  - Reduce samplers by looping (by thread and by controller), and use variables (CSV Data Set) to vary the sample
  - Use CSV output rather than XML