Politecnico
di Milano

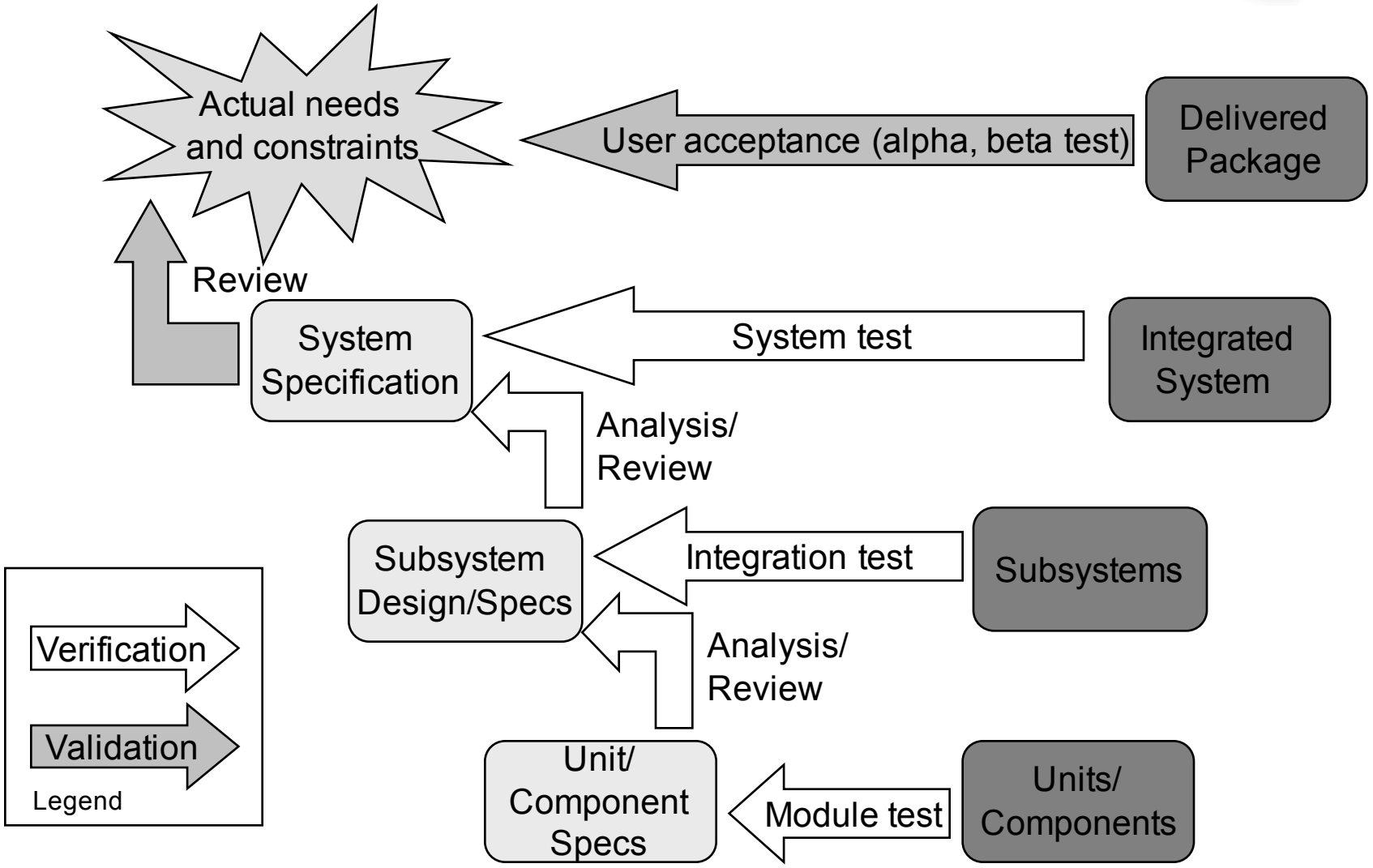
TestingTools

Michele Guerriero
michele.guerriero@polimi.it

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?"

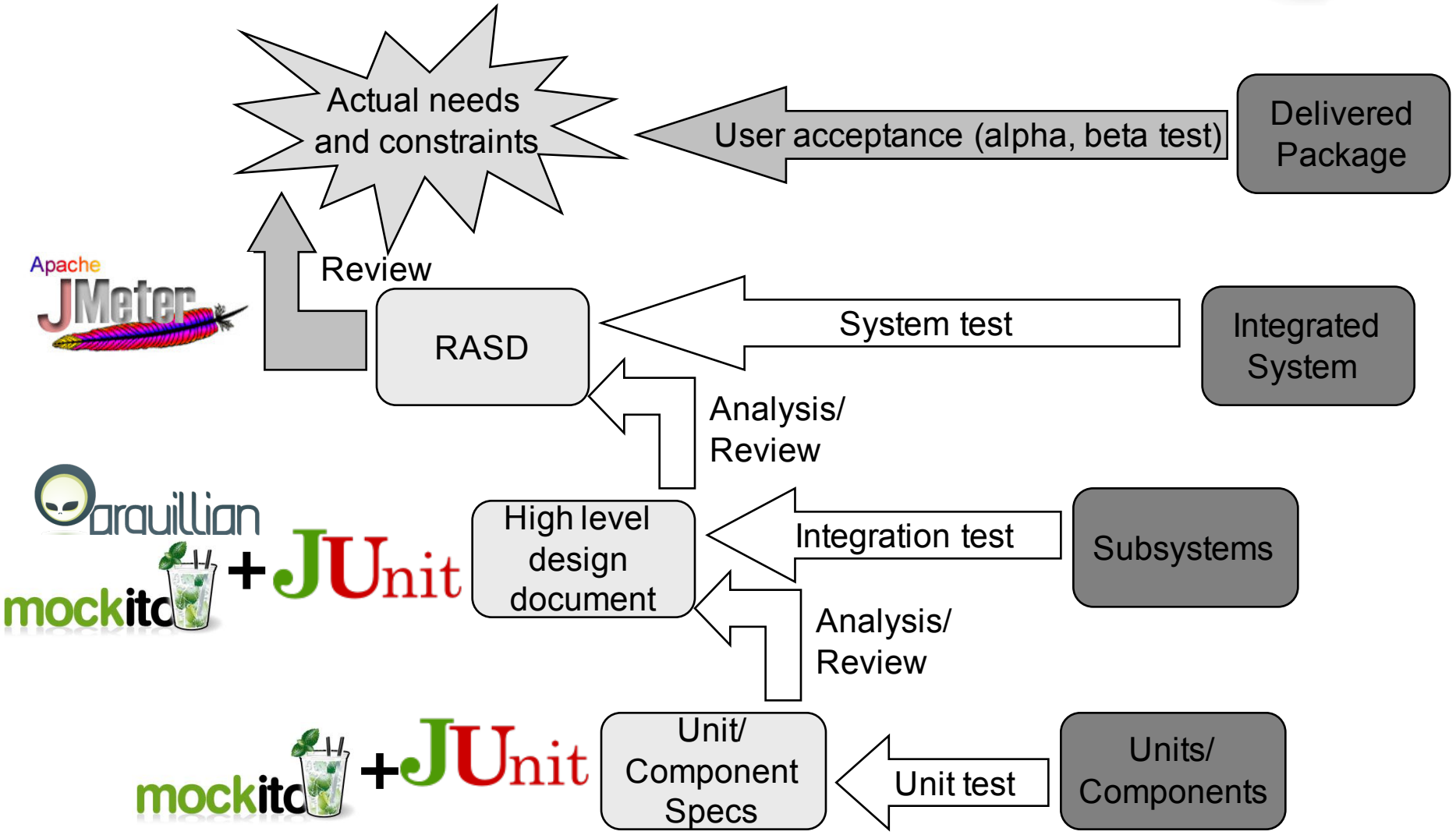
V&V activities and software artifacts (the V model)



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

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



Recall: types of requirements

- **Functional** requirements:
  - Are the main goals the software to be has to fulfill, the expected set of functionalities
  - Example:
    - A word processor user should be able to search for strings in the text
  - Different tools depending on the specific functionality and execution environment

- **Nonfunctional** requirements:
  - User visible aspects of the system not directly related to functional behavior
  - Examples:
    - The response time must be less than 1 second
    - The server must be available 24 hours a day
  - JMeter used for **performance testing**

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
- Supports the **scaffolding** by defining **stubs** when you cannot test a componet in isolation

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 verifying whether certain methods were called “n” times or never

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

Mockito - Example


```
@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();
}
```

Mocking a user of the AdditionBean (from the JEE classes)


```
@Test
public void test() {

    AdditionBean addBean = new AdditionBean ();
    MockedUser aUser= mock(MockedUser.class);

    when(aUser.getFirst()).thenReturn(43);
    when(aUser.getSecond()).thenReturn(57);

    addBean.setI(aUser.getFirst());
    addBean.setJ(aUser.getSecond());

    Assert.assertEquals(new Integer(100), addBean.getK());
}
```

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?

Arquillian tests

- An Arquillian test looks just like a **JUnit test, with some extra flair**.
- An Arquillian test case must have three things:
  - A **@RunWith(Arquillian.class)** annotation on the class; tells JUnit to use Arquillian as the test controller.
  - A public static method annotated with **@Deployment** that **returns a ShrinkWrap archive**; used to retrieve the **test archive** (i.e., **micro-deployment**) – see next slide.
  - At least one method annotated with @Test (a JUnit test); each @Test method is run inside the container environment.
  - The @Deployment method is only mandatory for tests that run inside the container.

What's a test archive?

- The purpose of the test archive is to **isolate the classes and resources** which are needed by the test from the remainder of the classpath
- You include only what the test needs (which may be the entire classpath, if that's what you decide)
- The archive is defined using **ShrinkWrap** which is a Java API for creating archives (e.g., jar, war, ear)
- The micro-deployment strategy lets you focus on precisely the classes you want to test
- Once the ShrinkWrap archive is deployed to the server, it becomes a real archive

Container Adapters

- After you write the test it is necessary to decide in **which container** (embedded or remote) it will be executed
- Arquillian selects the target container based on which **container adapter** is available on the test **classpath**
- A *container adapter* controls and communicates with a container (e.g. JBoss AS, GlassFish, etc)
- An Arquillian test can be executed in any **container that's compatible with the programming model used** in the test (i.e. **JEE**)
- To switch to another container, you just change which container adapter is on the classpath before you run the test.

Testing the injection of the AdditionBean EJB (from the JEE classes)

```
@RunWith(Arquillian.class)
public class AdditionBeanTest {

    @EJB
    private AdditionBeanBasic addBean;

    @Deployment
    public static WebArchive createDeployment() {
        return ShrinkWrap.create(WebArchive.class)
                .addClasses(AdditionBeanBasic.class, IntPair.class)
                .addAsManifestResource(EmptyAsset.INSTANCE, "beans.xml")
                .addAsResource("test-persistence.xml", "META-INF/persistence.xml");

    }

    @Test
    public void additionBeanInjectionTest() {
        addBean.setI(Integer.parseInt("3"));
        addBean.setJ(Integer.parseInt("4"));
        addBean.add();
        Assert.assertEquals(new Integer(3), addBean.getI());
        Assert.assertEquals(new Integer(4), addBean.getJ());
        Assert.assertEquals(new Integer(7), addBean.getK());
    }

}
```

From ArquillianAPIs

From JEE APIs

From ArquillianAPIs

From JUnit APIs

Jmeter: a load testing tool for analyzing and measuring performance

http://jmeter.apache.org

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
- Build and run most types of tests, e.g. Web (HTTP/HTTPS), FTP, JDBC,
- 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

From Jendrock et al. *The Java EE 7 Tutorial*

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

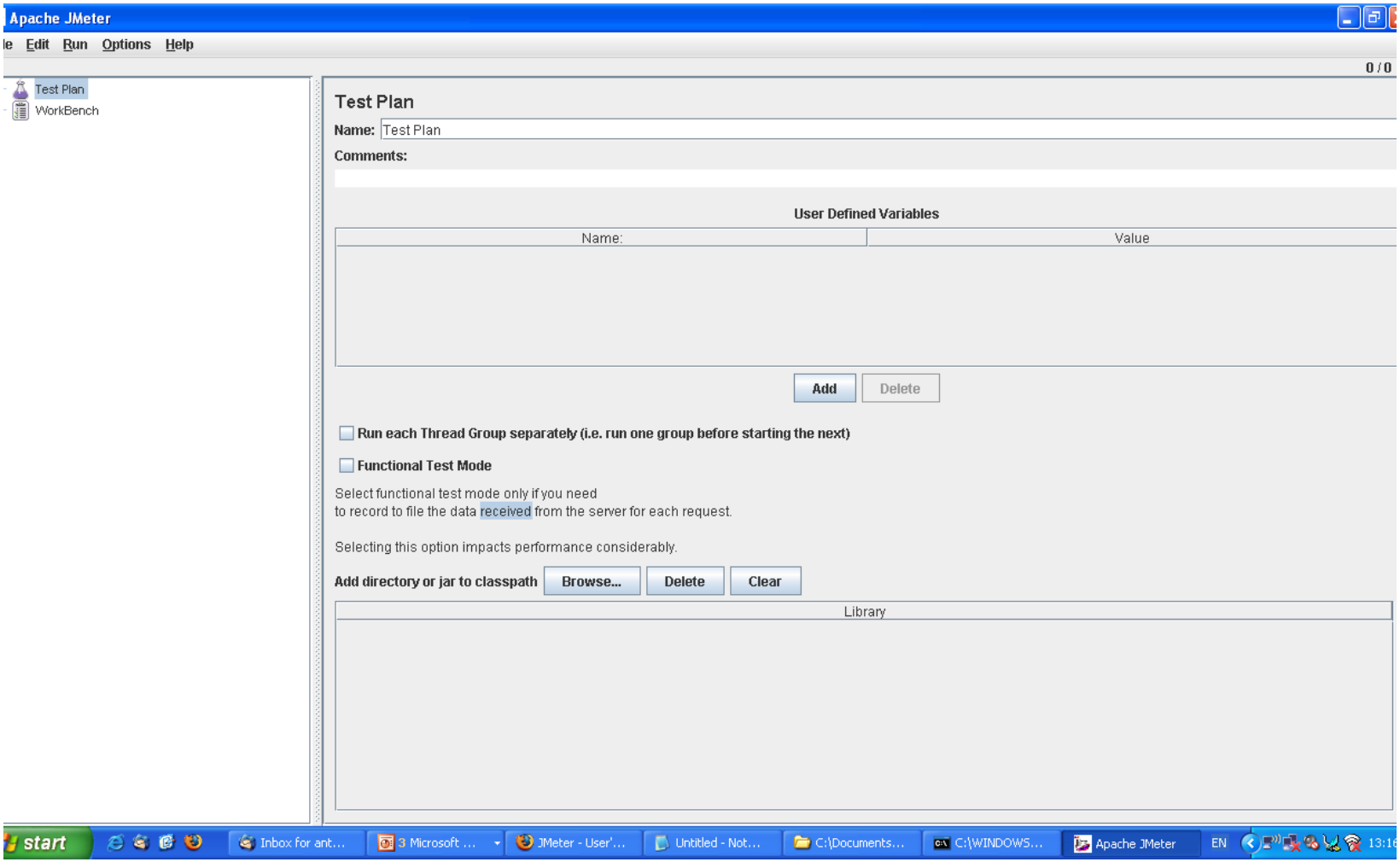
What Can You Do With It?

- JMeter lets you set up test plans that simulate logging into a web site, filling out forms, clicking buttons, links, etc.
- A test plan describes a series of steps JMeter will execute when run. A complete test plan will consist of one or more Thread Groups, logic controllers, sampler controllers, listeners, timers, etc.
- You can simulate the number of users doing this, the rate that they do it…

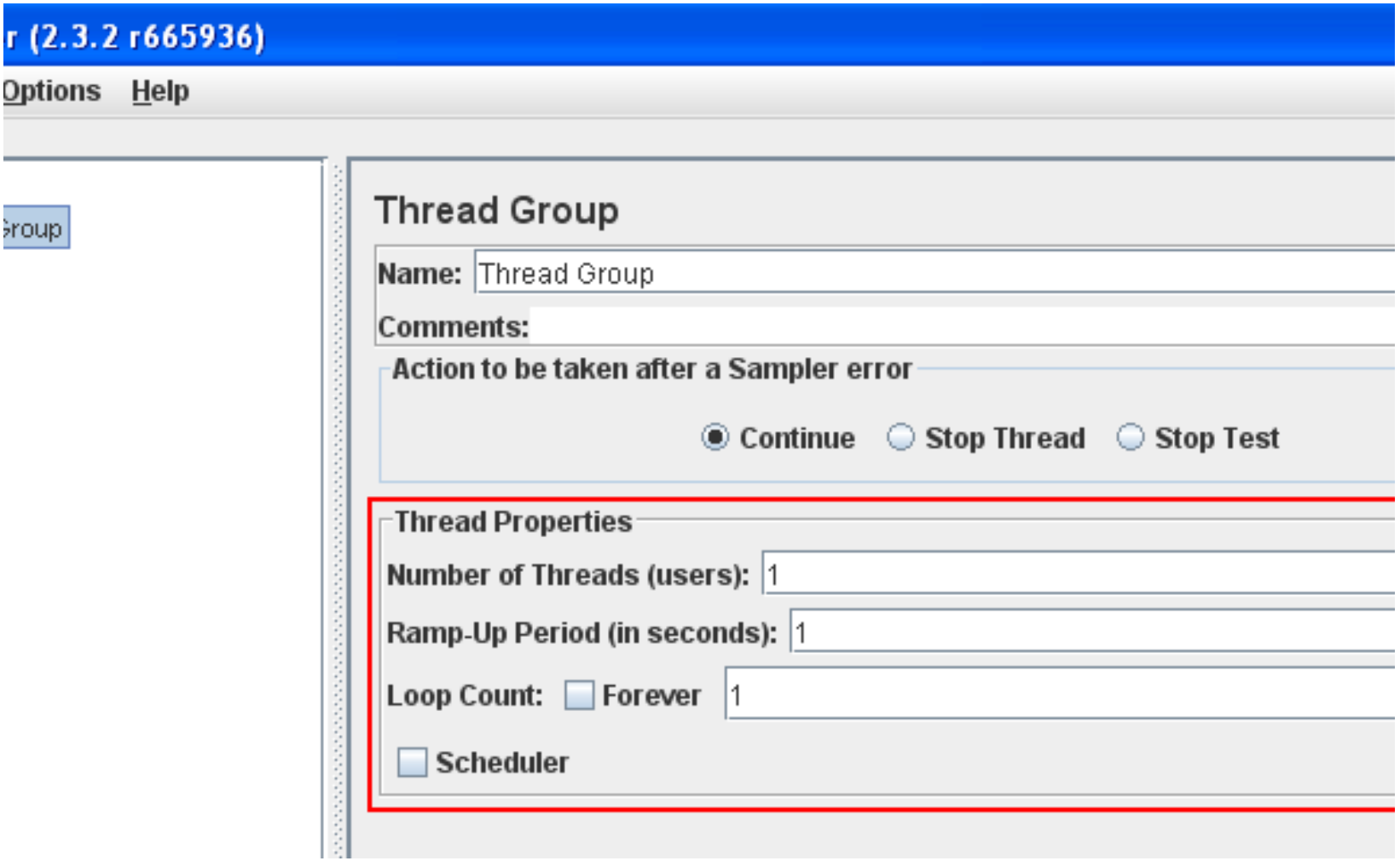
Terminologies used in J-Meter

- Thread Group: tells JMeter the number of users you want to simulate, how often the users should send requests, and how many requests they should send
- Number of Threads: Number of virtual users
- 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

The test plan

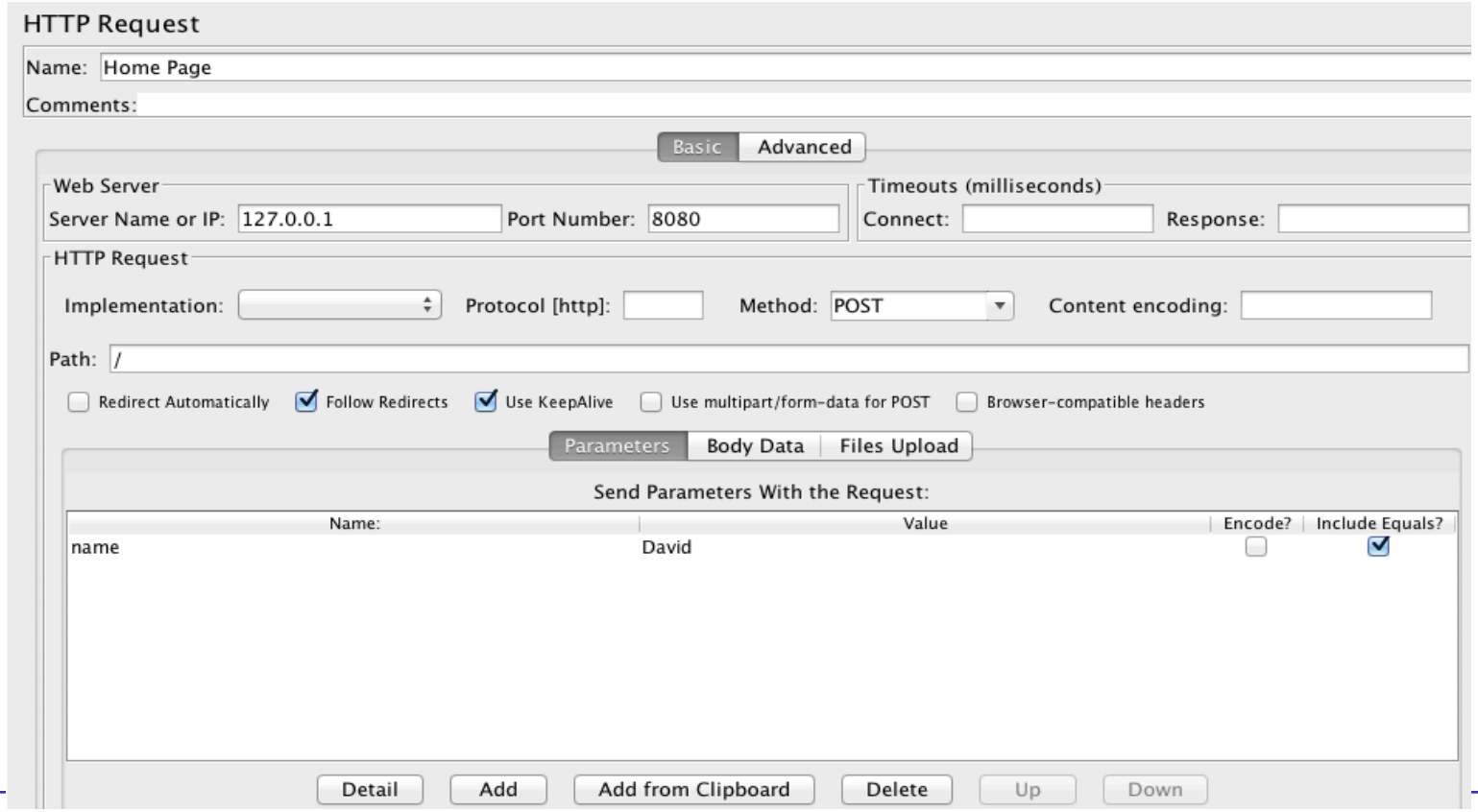


The thread group



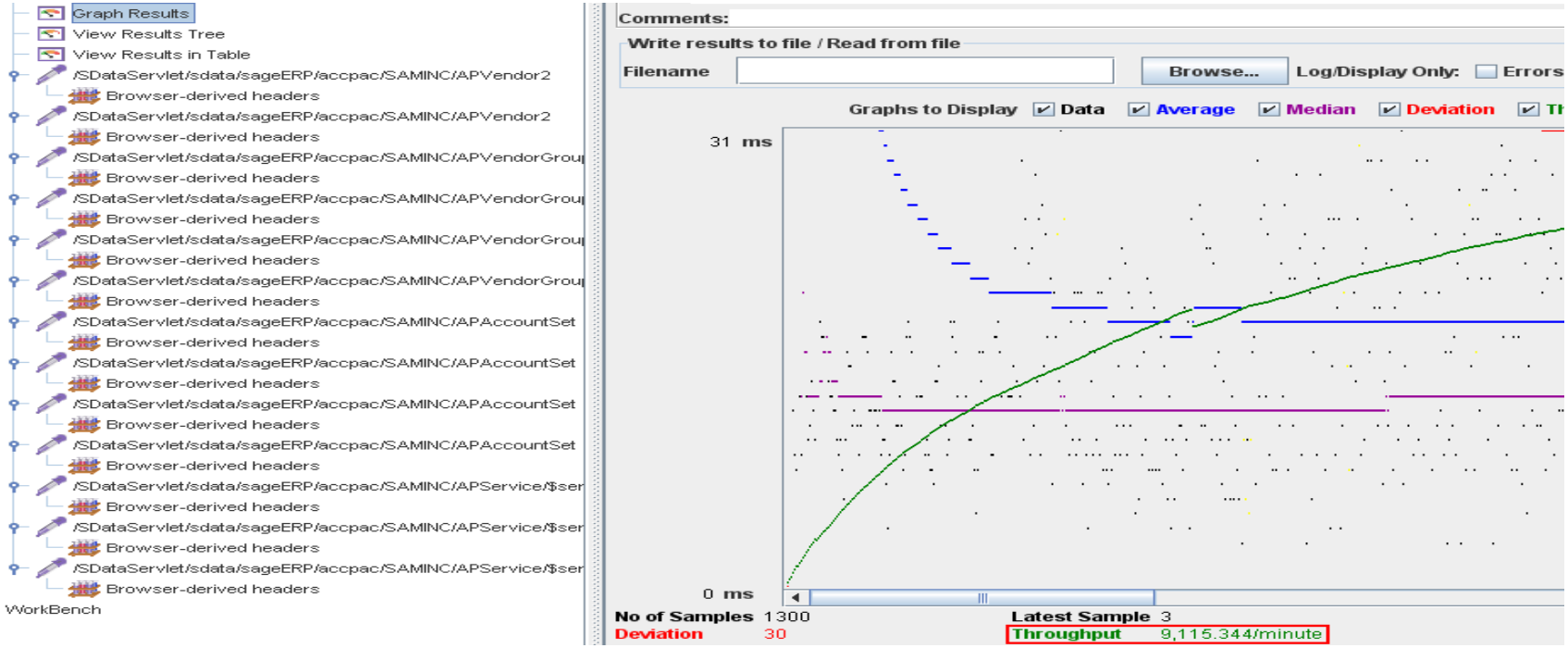
A Sampler Controller: the HTTP Request

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



Listeners

- 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



Logic Controllers

- 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. For example, you can add an Interleave Logic Controller to alternate between two HTTP Request Samplers.

- 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)