

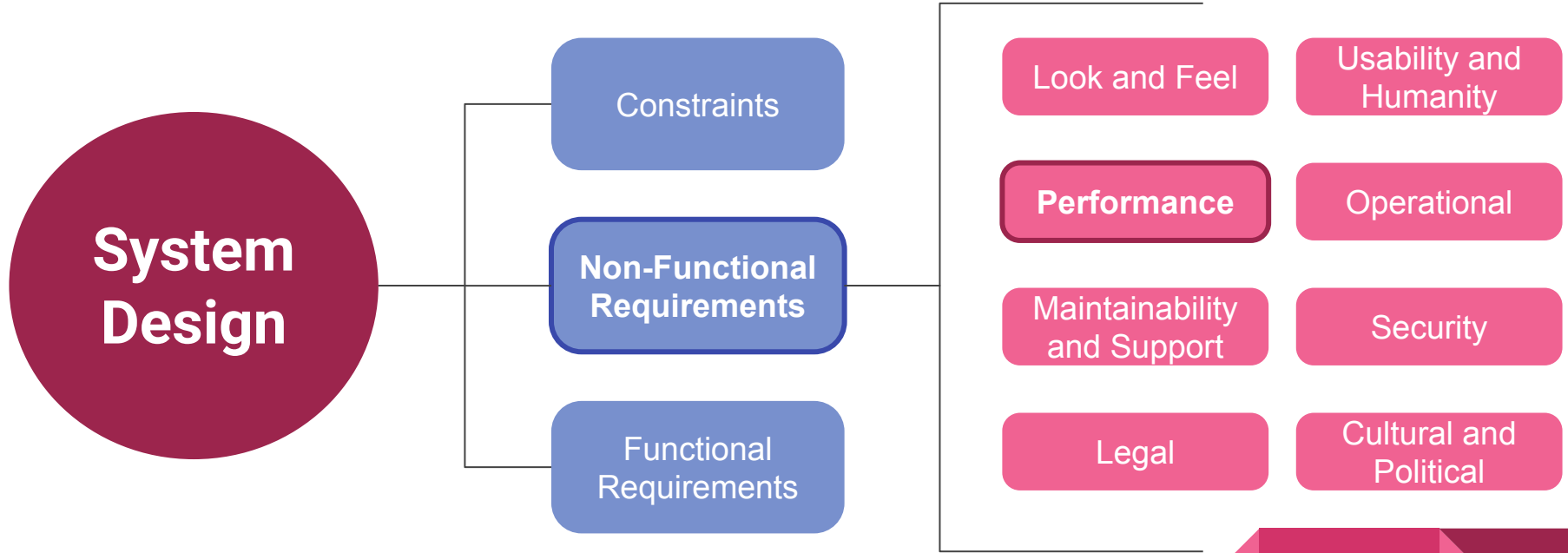
Performance Testing

Gonçalo Marantes - up201706917@edu.fe.up.pt

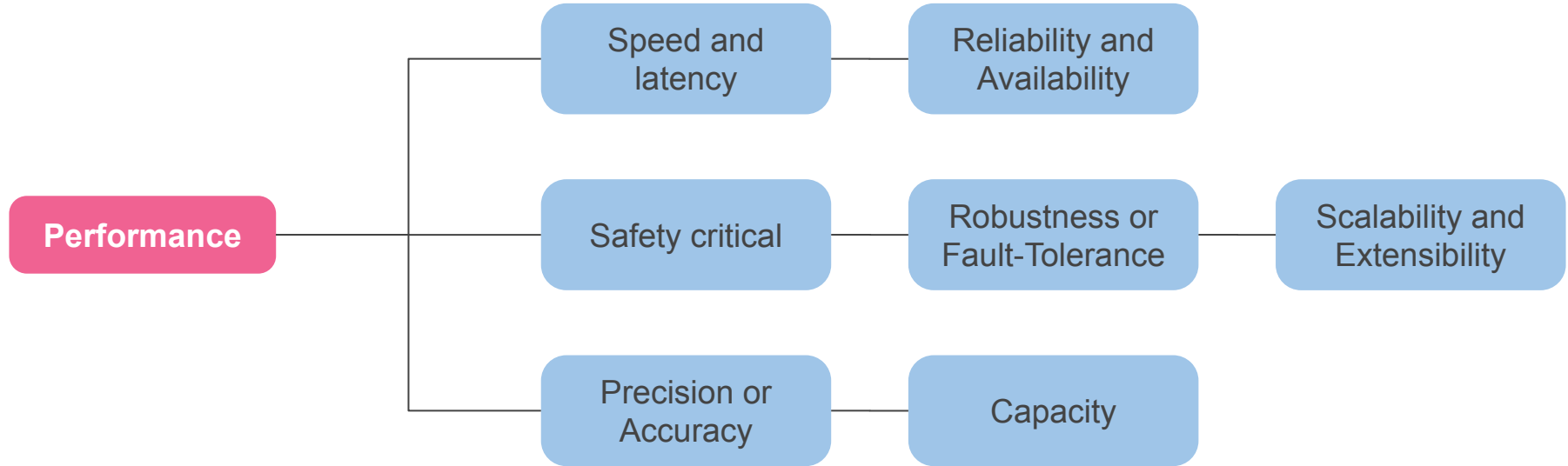
Ricardo Cardoso - up201604686@edu.fe.up.pt

Tiago Silva - up201705985@edu.fe.up.pt

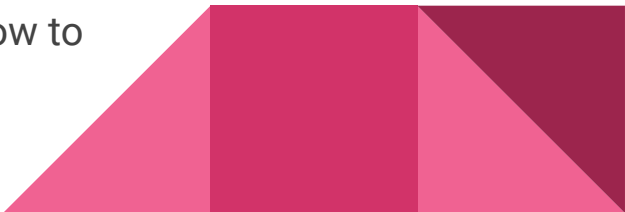
Performance in Software Systems Design



Types of Performance Requirements



Examples of Performance Requirements

- **Speed and Latency:** “The response shall be fast enough to avoid interrupting the user's flow of thought”
 - **Precision or Accuracy:** “All monetary amounts shall be accurate to 2 decimal places.”
 - **Safety Critical:** “The system shall not discriminate based on ethnicity”
 - **Reliability and Availability:** “The system shall achieve 99% up time.”
 - **Robustness or Fault-Tolerance:** “The product shall continue to operate in local mode whenever it loses its link to the central server.”
 - **Capacity:** “The product shall cater for 300 simultaneous users within the period from 9:00 A.M. to 11:00 A.M. Maximum loading at other periods will be 150 simultaneous users.”
 - **Scalability or Extensibility:** “The system shall be capable of processing the existing 100,000 customers. This number is expected to grow to 500,000 within three years.”
- 

Why do we need Performance Testing?

- When designing a system it is important to **validate the system's performance** by providing **performance testing results to stakeholders**
- **Test** the non-functional **performance** requirements in a **real-world and production-ready environment**.
- By doing performance testing we can identify (a priori):
 - Bottlenecks
 - Capacity cap
 - Availability
 - Scalability
 - System behaviour in an “hostile” environment
 - ...
- Better safe than sorry! ^_(\ツ)_/^-



What is Performance Testing?

Performance Testing is a software testing process used to assess how an application performs **under a determined workload**.

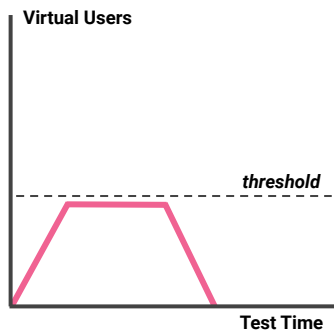
Ultimately, performance testing, aims to gauge whether the aforementioned non-functional requirements related with speed, response time, stability, reliability, scalability and resource usage are met.



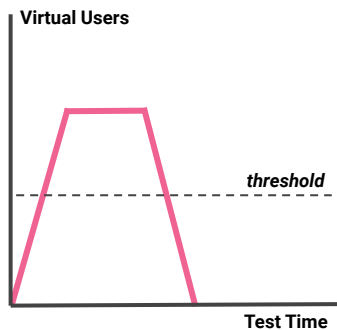
Types of Performance Testing

There are 4 main types of performance testing, each one looking to test a different real-world situation:

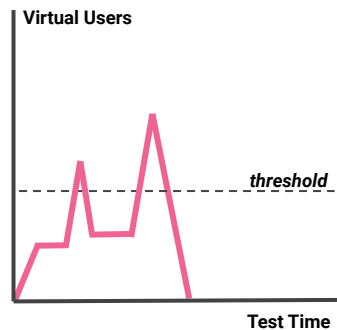
Load Tests



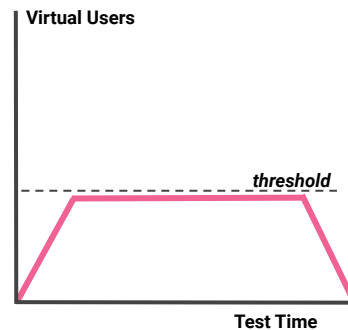
Stress Tests



Spike Tests



Endurance Tests



Some add more types of tests to this list but are a little more specific

- Breakpoint Tests
- Volume Tests
- Scalability Tests

What metrics are important? It depends...

During the execution of the tests some **measurements** are taken that are used to calculate **Performance metrics**.

Some performance metrics:

- Requests per second
- Average Response Time
- Peak Response Time
- Error Rate
- Wait time
- Concurrent users
- Throughput
- Total user sessions
- CPU utilization
- Memory utilization



How to Performance Test

Elicit non-functional requirements

In the beginning (and during) of the project

Configure the environment as close as possible to the production environment

Compare with the previously defined values



Define desired threshold values based on the non-functional requirements

Tools Comparison



	Gatling	JMeter	LOCUST.io	TSUNG	WEB LOAD	The Grinder
Open source	Yes	Yes	Yes	Yes	No (Paid)	Yes
OS	All	All	All	Linux/Unix	All	All
GUI	Recorder only	Full	No	No	Full	Console only
Test Recorder	HTTP	HTTP	No	HTTP, Postgres	HTTP	TCP (including HTTP)
Test Language	Scala	XML	Python	XML	XML	Python, Clojure
Extension Language	Scala	Java, Beanshell, Javascript, Jexl	Python	Erlang	Javascript	Python, Clojure
Load Reports	HTML	XML, CSV, Tables, Graphs	HTML	HTML	JSON, Embedded Tables, Graphs, Web	Console
Protocols	HTTP, JDBC, JMS	HTTP/HTTPS, XML, SOAP, TCP	HTTP/HTTPS, WebSocket, etc.	HTTP, WebDAV, Postgres, MySQL, XMPP, WebSocket, AMQP, MQTT, LDAP	HTTP/HTTPS, WebSocket, PUSH, AJAX, SOAP, HTML5, WebDAV	HTTP, SOAP, JDBC, POP3, SMTP, LDAP, JMS
Host Monitoring	No	Yes w/ PerfMon plugin	No	Yes	Yes	No

Tools Comparison



	Gatling	JMeter	LOCUST.io	TSUNG	WEB LOAD	The Grinder
Open source	Yes	Yes	Yes	Yes	No (Paid)	Yes
OS	All	All	All	Linux/Unix	All	All
GUI	Recorder only	Full	No	No	Full	Console only
Test Recorder	HTTP	HTTP	No	HTTP, Postgres	HTTP	TCP (including HTTP)
Test Language	Scala	XML	Python	XML	XML	Python, Clojure
Extension Language	Scala	Java, Beanshell, Javascript, Jexl	Python	Erlang	Javascript	Python, Clojure
Load Reports	HTML	XML, CSV, Tables, Graphs	HTML	HTML	JSON, Embedded Tables, Graphs, Web	Console
Protocols	HTTP, JDBC, JMS	HTTP/HTTPS, XML, SOAP, TCP	HTTP/HTTPS, WebSocket, etc.	HTTP, WebDAV, Postgres, MySQL, XMPP, WebSocket, AMQP, MQTT, LDAP	HTTP/HTTPS, WebSocket, PUSH, AJAX, SOAP, HTML5, WebDAV	HTTP, SOAP, JDBC, POP3, SMTP, LDAP, JMS
Host Monitoring	No	Yes w/ PerfMon plugin	No	Yes	Yes	No

LOCUST.io

Write test plan in Python

- You don't need to slowly click on the UI interface, just write code.
- Locust is based on coroutines rather than callbacks, which makes your code execute synchronously similar to normal Python blocking code.

Distributed & scalable

- It supports hundreds of thousands of user behavior simulations.
- Locust supports running load tests on multiple computers (multiple machines can be run in parallel).
- Because of being event based, even a locust node can handle thousands of users in a single process.
- But even if you simulate so many users, not everyone is using your system at this frequency. Usually, when users are using a system, they spend time thinking about what to do next.
 - **The number of requests per second is not equal to the number of online users.**

```
import time
from locust import HttpUser, task, between

class QuickstartUser(HttpUser):
    wait_time = between(1, 5)

    @task
    def hello_world(self):
        self.client.get("/hello")
        self.client.get("/world")

    @task(3)
    def view_items(self):
        for item_id in range(10):
            self.client.get(f"/item?id={item_id}", name="/item")
            time.sleep(1)

    def on_start(self):
        self.client.post("/login", json={"username": "foo", "password": "bar"})
```

LOCUST.io

Statistical results based on web interface

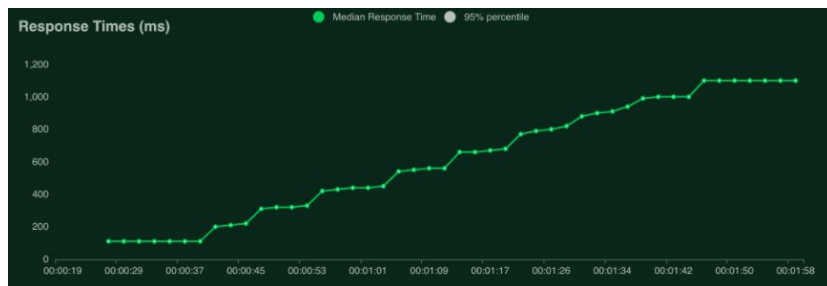
- Locust has a simple user interface to display relevant test details in real time.
- The statistical results interface is based on Web pages, so locust is cross platform and easy to expand.

Can test any web page / application / system

- Even if locust is web oriented, it can test almost any project
- Just use Python to write the scheme you want to test, and then implement a locust client to connect the project you want to test. It's very simple!

Disadvantages

- If you don't know much about python, it may not be so simple.
- If you are ignorant of programming, you should go back to JMeter.



References

- Apache Jmeter - <https://jmeter.apache.org/>
- Download WebLOAD - <https://www.radview.com/webload-download/>
- Gatling - <https://gatling.io/docs/gatling/>
- Locust Tutorial - <https://develloppaper.com/locust-tutorial/>
- Locust Documentation - <https://docs.locust.io/en/latest/index.html>
- Pohl, K. & Rupp, C. (2015). *Requirements Engineering Fundamentals* (2nd ed.) Rocky Nook
- The Grinder, a Java Load Testing Framework - <http://grinder.sourceforge.net/index.html>
- Tsung - <http://tsung.erlang-projects.org/1/01/about/>
- Volere Requirements Specification Template - <https://www.volere.org/templates/volere-requirements-specification-template>

