




# IS2545, LECTURE 16: PERFORMANCE TESTING 1

Bill Laboon



BEFORE WE TALK PERFORMANCE  
TESTING...

LET'S ASK: WHAT DO WE MEAN BY  
**PERFORMANCE?**



“I shall not today attempt further to define the kinds of material I understand to be embraced within that shorthand description, and perhaps I could never succeed in intelligibly doing so. But ***I know it when I see it.***”

-Potter Stewart, US Supreme Court Justice  
Concurrence in *Jacobellis v. Ohio*



BUT...

- We *can* talk about what kinds of performance we're interested in, and set specific goals for the system under test.
- A video game console will have very different performance requirements than a weather forecasting supercomputer.

# PERFORMANCE INDICATORS

- Quantitative measures of the performance of a system under test
- Examples:
  - How long does it take to respond to a button press?
  - How many users can access the system at one time?
  - How long can the system go without a failure?
  - How much CPU does a standard query on the database use?
  - How big is the program in megabytes?
  - How fast the program calculate some function?

# KINDS OF PERFORMANCE INDICATORS

- Service-Oriented
- Efficiency-Oriented



## SERVICE-ORIENTED

Service-oriented indicators measure how well a system is providing a service to the users. They are measured from a specific user's point of view.



# EFFICIENCY-ORIENTED

Efficiency-oriented indicators measure how well the system makes use of the computational resources available to it. This is looking at the performance from a more developmental perspective.





# CATEGORIES OF SERVICE-ORIENTED INDICATORS

- Availability - How available is the system to the user?  
What percentage of the time can they access it?
- Response Time - How quickly does the system  
respond to user input?



## CATEGORIES OF EFFICIENCY-ORIENTED INDICATORS

- Throughput - How many events can occur and be processed in a given amount of time?
- Utilization - What percentage or absolute amount of computing resources are used to perform a task?

# TESTING PERFORMANCE

- In order to test performance, you should have **performance targets** - quantitative values that the **performance indicators** should reach.
- Performance targets may be assigned to a subset of the most important performance indicators, **called key performance indicators (KPIs)**.

# PERFORMANCE THRESHOLDS

- **Performance thresholds** are the absolute minimum performance level a system can reach and be considered production-ready.
- These may be used in addition to performance targets as a “bare minimum” to reach, although not the desired target.

# EXAMPLE


- You are developing a web application which is expecting a relatively constant 20 hits per second.
- A key performance indicator (KPI) might be response time, with a performance threshold of three seconds mean time to respond and a performance target of one second.
- Ideally, your stakeholders would like sub-second response time, but they would be satisfied as long as the response time is below three seconds.

# TESTING SERVICE-ORIENTED PERFORMANCE INDICATORS – RESPONSE TIME

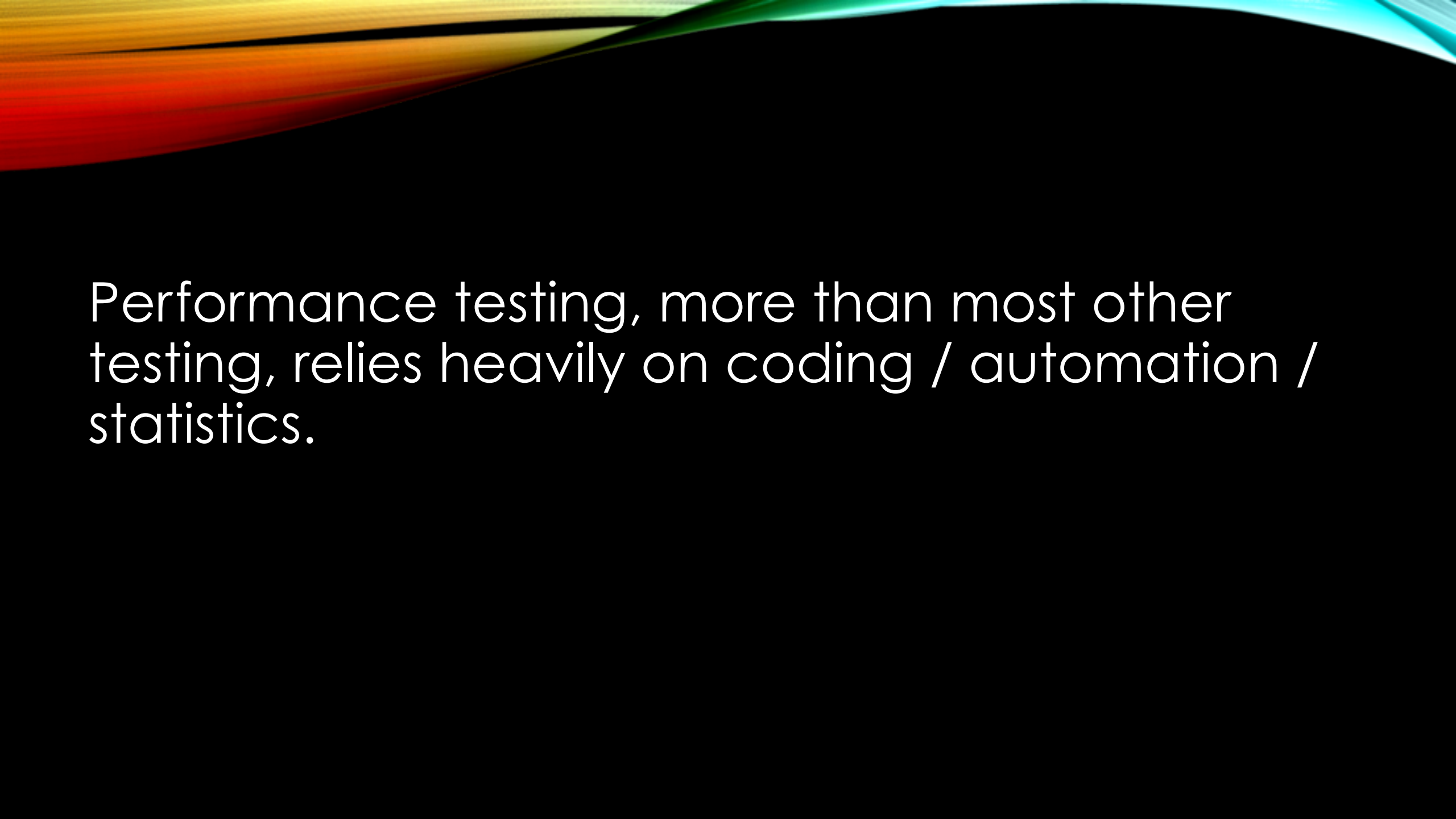
- Response time
- Easy to do!
  - Do something
  - Click “start” on stopwatch
  - Wait for response
  - Click “stop” on stopwatch
  - Write down number on stopwatch!



Any problems with this approach?

- 
1. Impossible to measure sub-second response times
  2. Human error
  3. Probably the most boring thing a person can do
  4. Time-consuming
  5. Impossible to measure "hidden" responses
  6. Inability to get large data sets





Performance testing, more than most other testing, relies heavily on coding / automation / statistics.

# STATISTICS? WHY?

- You shouldn't really trust a single result in performance testing, especially for response times. You should always be trying multiple times and discussing a mean value, maximum value, minimum values, etc.
- There are so many variables in a single test run (other processes taking up CPU, pipelining issues, memory swaps, VM startup times, etc.) that a single test run is almost worthless.



# HUMAN ERROR

- Minor issues with human error can cause massive changes in performance results.

# SHE BLINDED ME WITH SCIENCE

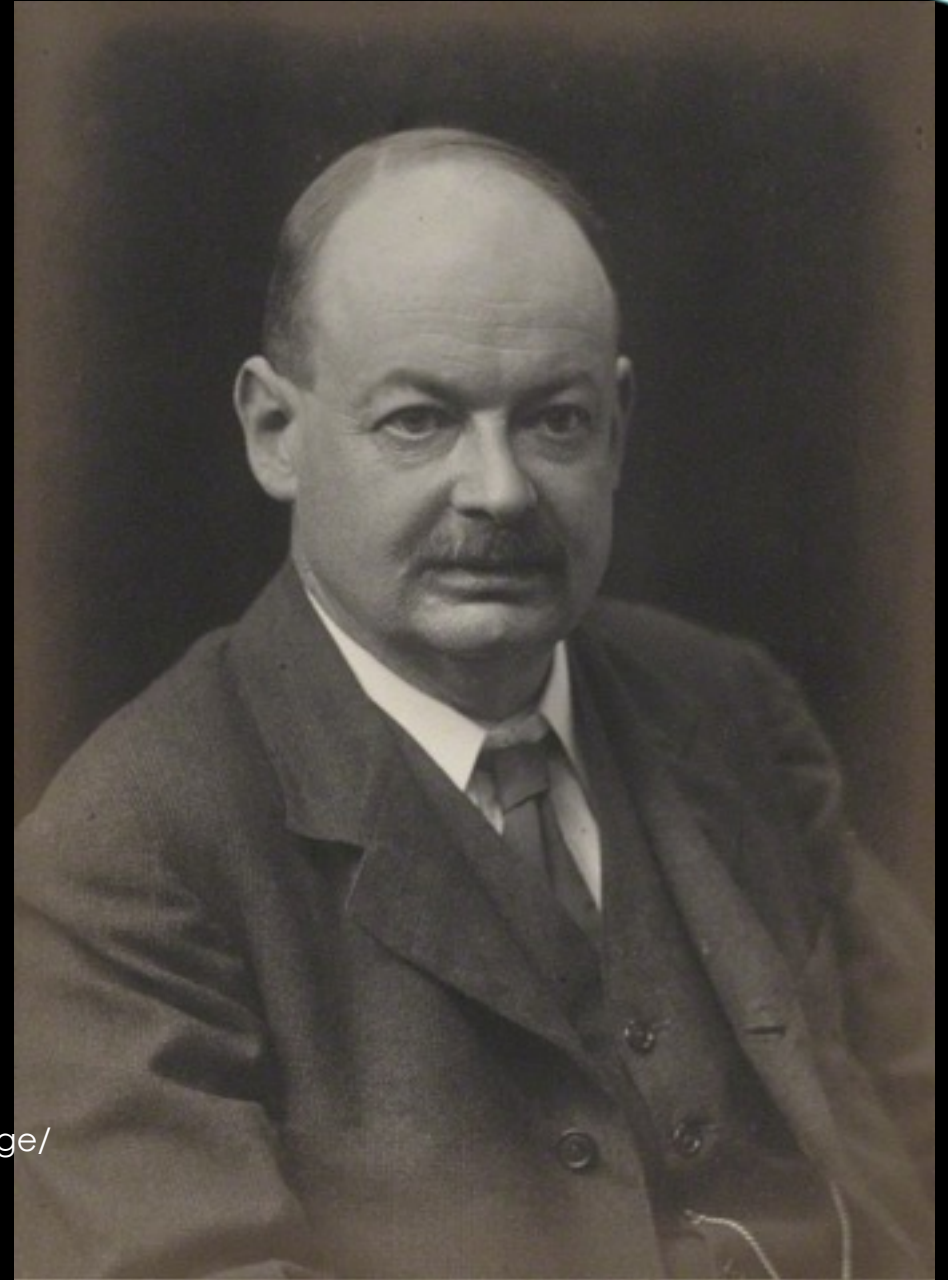
- Performance testing is more like a science than other kinds of testing.
- You want to run multiple experiments and eliminate all the other variables OTHER THAN THE CODE UNDER TEST.
- Don't run version 1 on a ChromeBook and version 2 on a Cray supercomputer and talk about the massive speed improvement.

# KINDS OF EVENTS TO TEST FOR RESPONSE TIME

- Time for calculation to take place
- Time for character to appear on screen
- Time for image to appear
- Time to download
- Time for server response
- Time for page to load
- Time for code to execute

# WHAT IS TIME?

By Walter Stoneman - <http://www.npg.org.uk/collections/search/portraitLarge/mw124669/John-McTaggart-Ellis-McTaggart>, Public Domain, <https://commons.wikimedia.org/w/index.php?curid=22142173>




# ASSUMING THAT TIME EXISTS...

- user time: Amount of time user code executes
- system time: Amount of time kernel code executes
- total time : user time + system time
- real time: “Actual” amount of time taken (wall clock time)

# EXAMPLE

- time command in Unix
  - time java Foo
  - time curl <http://www.example.com>
  - time ls -l
- Windows PowerShell has something similar
  - Measure-Command { java Foo -wait }



- 
- Users almost always care about real (“wall clock”) time.
  - You usually use total, user, and system time to help developers, not as KPIs in and of themselves.

# ROUGH OUTLINE FOR RESPONSE TIME PERFORMANCE TARGETS

- < 0.1 S : Response time required to feel that system is instantaneous
  - < 1 S : Response time required for flow of thought not to be interrupted
  - < 10 S : Response time required for user to stay focused on the application (and not go re-load Reddit)
- taken from Usability Engineering by Jakob Nielsen, 1993

*Things haven't changed much since then!*

# MEASURING AVAILABILITY PERFORMANCE INDICATORS

- Availability - often referred to as uptime - how often can a user access the system when they expect to be able to do so?
- Often referred to as a SLA (service-level agreement).
  - “I am a web host. I guarantee you that you and your users will be able to access your server 99% of the time in a given month.”

# NINES

- Uptime is often expressed in an abbreviated form as 9's (e.g. 3 nines, 5 nines etc)
- Refers to how many 9's start out the percentage of time available
  - 1 nine: 90% available (36.5 days of downtime per year)
  - 2 nines: 99% available (3.65 days of downtime per year)
  - 3 nines: 99.9% available (8.76 hours of downtime per year)
  - 4 nines: 99.99% available (52.56 minutes of downtime per year)
  - 5 nines: 99.999% available (5.26 minutes of downtime per year)
  - 6 nines: 99.9999% available (31.5 seconds of downtime per year)
  - 9 nines: 99.99999999% available (31.5 ms of downtime per year)



# HOW TO TEST?

- Often difficult – most managers won't let you run a few “test years” before deploying it for real
- Modeling the system and estimating uptime is the best (feasible) approach

# DETERMINE VALUES FOR MODEL WITH LOAD TESTING

- **Load testing** - how many concurrent users and/or work can the system handle?
- Kinds of load testing:
  - **Baseline Test** - A bare minimum amount of use, to provide a baseline
  - **Soak / Stability Test** - Leave it running for an extended period of time, usually at low levels of usage
  - **Stress Test** - High levels of activity



# REALITY

- For true availability numbers, also need to determine:
  - Likelihood of hardware failure
  - Likelihood of program-ending bugs
  - Planned maintenance
  - etc.

# SIC TRANSIT GLORIA MUNDI

- Even with all this work, things go wrong
- Many major service providers, such as Microsoft Azure and Amazon Web Services, “breach” their SLAs in a given month (e.g. their servers are guaranteed to be available 99.9% of the time, but are down for two days due to a power surge, meaning they were only available 93.6% of that month)
- Usually, money is refunded automatically



# DEVELOPING THE SERVICE-ORIENTED TEST PLAN

- Think from a user's perspective!
  - How fast do I expect this to be?
  - What things matter to me, speedwise?
  - How often do I expect this to be available?
  - Are large variances in response time allowed?

# DETERMINE KPIS, TARGETS, AND THRESHOLDS

- Example:
  - Average page load time – Target: less than two seconds, Threshold: less than five seconds
  - Max page load time – Target: less than five seconds, Threshold: less than ten seconds
  - Availability of system: Target: greater than 99.9%, threshold: greater than 99%



# THINK ABOUT CONTINGENCY PLANS!

- What if performance requirements aren't met?
- What if they can't be?
- What if they can be, but at a high cost in time/  
resources?
- etc.