STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

SSW-567: Software Testing, Quality Assurance, and Maintenance

Performance Testing
*Reliability Testing**

Prepared by Raz Saremi
Presented by Lu Xiao
Software Engineering
School of Systems and Enterprises

*This lecture is based on content originally prepared by Linda Laird, and includes updates by Jim Rowland and Rich Kempinski

Today's topics

Performance testing

- Types of performance tests
- Why it is important
- What do we need for performance testing

Reliability testing

- Software failure characteristics
  - Not all defects are the same
- Usage-based or statistical testing
- Operational Profiles

Performance Testing

- Recall our definition of Software Testing…

  A technical Investigation of the System Under Test conducted to provide the Stakeholders with Quality-related information

- What quality-related information are we are looking for?
- What would stakeholders care about?
- What are some of the things that could go wrong?
- What are Faults? Failures?
- What are the Performance Requirements?

Performance, Stress, Load Testing

Test Type	Description
Performance	Conformance to performance requirements
Stress	Performance at and beyond expected loads
Load	Performance characteristics under varied loads
Soak/ Endurance	Performance characteristics over long periods of time

Performance requirements are needed ***before*** Performance and Stress testing

These may all be grouped into the term "performance testing"

What Are We Testing?

Performance Testing

Conformance to **response/execution time**, **throughput** and **capacity** requirements

Stress Testing

System behavior **at and beyond** the system's limits

Does performance **degrade** gracefully or fail catastrophically?

Load Testing

How does the system perform under different, **specific loads**?

Identify **performance bottlenecks**

Soak Testing/Endurance Testing

How does the system perform under **continuous expected load**?

Detect memory leaks, system resources, etc.

Common Performance Problems

Poor stability

The application is unstable under load over time

Poor response time

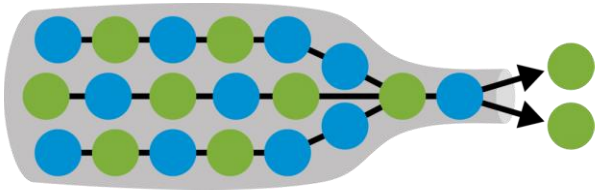
User experience delays

Poor scalability

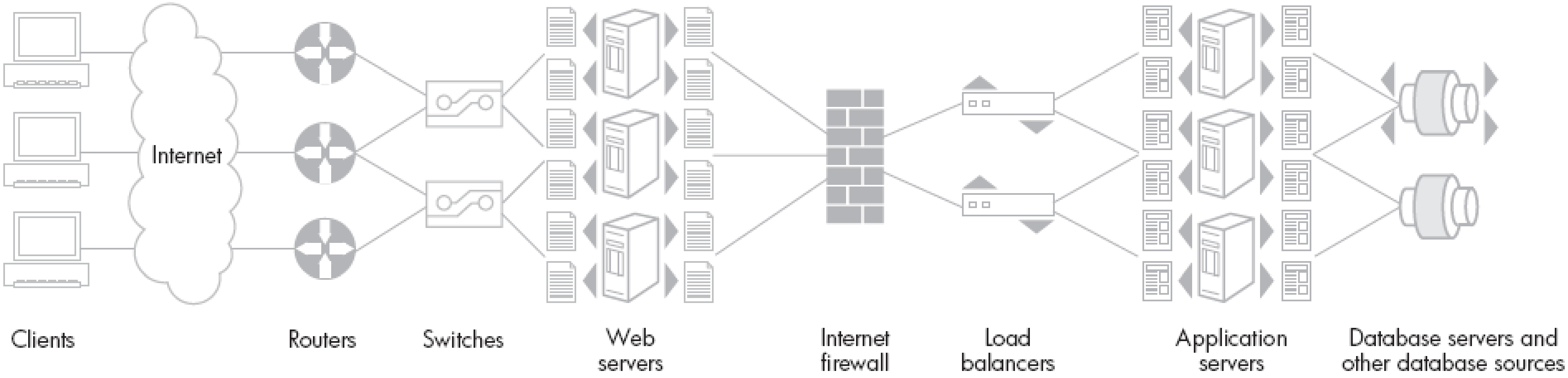
The application doesn't scale to the desired number of users

Bottlenecks

Performance barriers, e.g. CPU, Memory, Network, OS, Disk



Consider this complex web architecture…



Will the system's performance meet the requirements?

How should we tune it to improve and meet customer expectations?

Source: HP White Paper – Intro to Load Testing

Performance Testing Goals

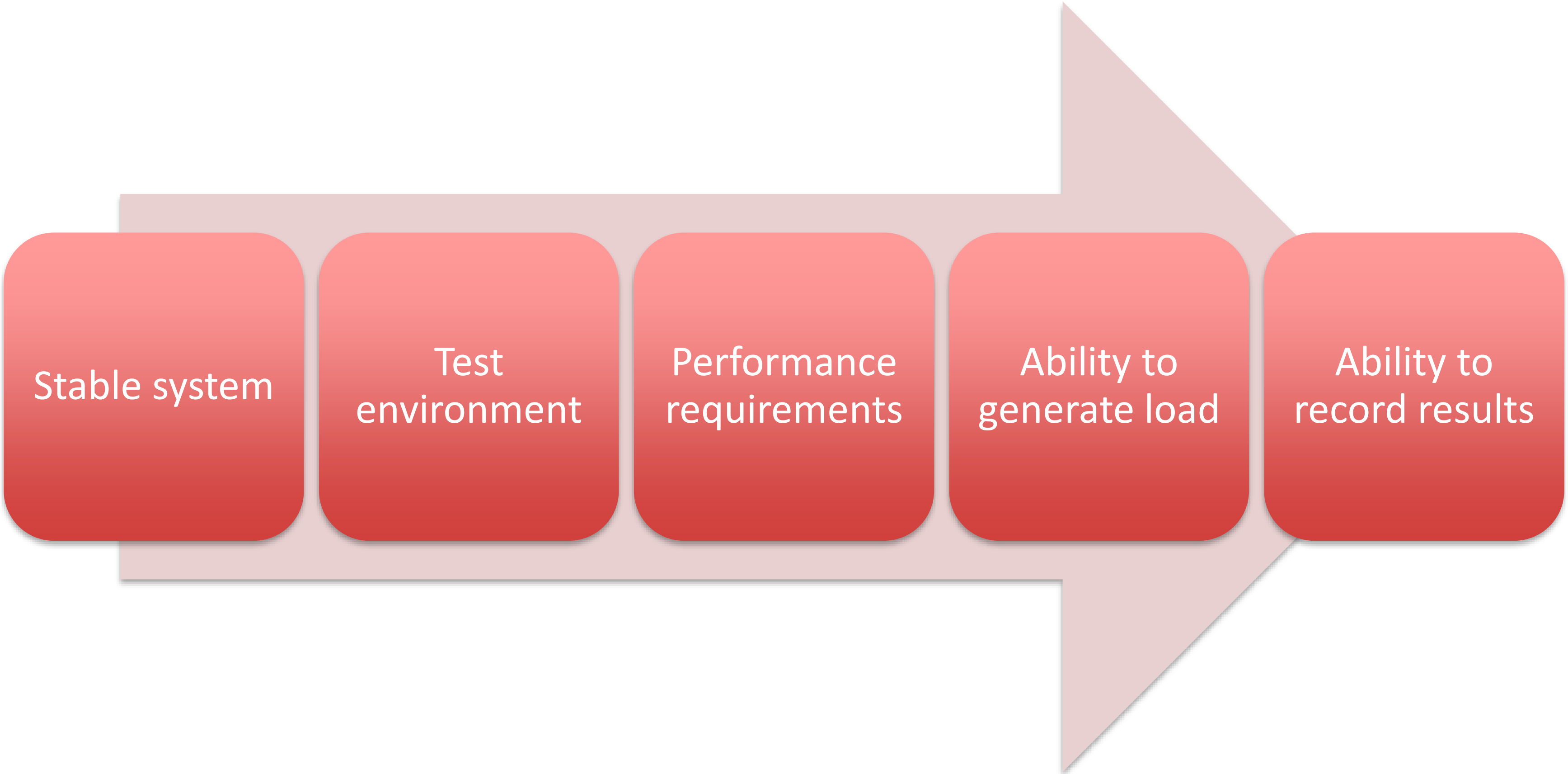
Ensure that the application is operating with:

- Optimum responsiveness
- Scalability
- Throughput
- Stability

To achieve these goals, we must:

- Measure & benchmark performance
- Identify performance bottlenecks
  - Measure usage of system resources
- Utilize testing results & metrics to identify how to tune the application configuration for optimal use of system resources
- Understand system limitations and degradation behaviors

What’s needed for Performance Testing?



Stable system

Test environment

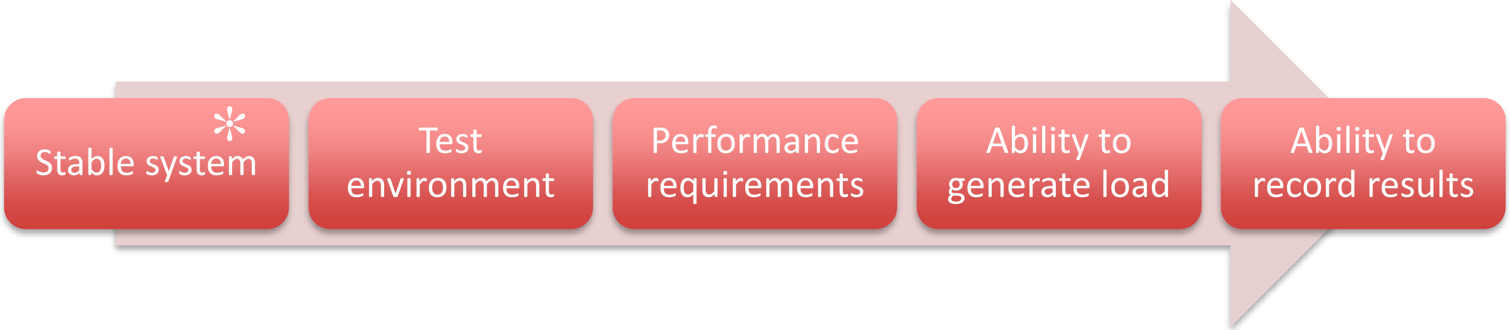
Performance requirements

Ability to generate load

Ability to record results

and, talented performance testers…

What’s needed for Performance Testing?



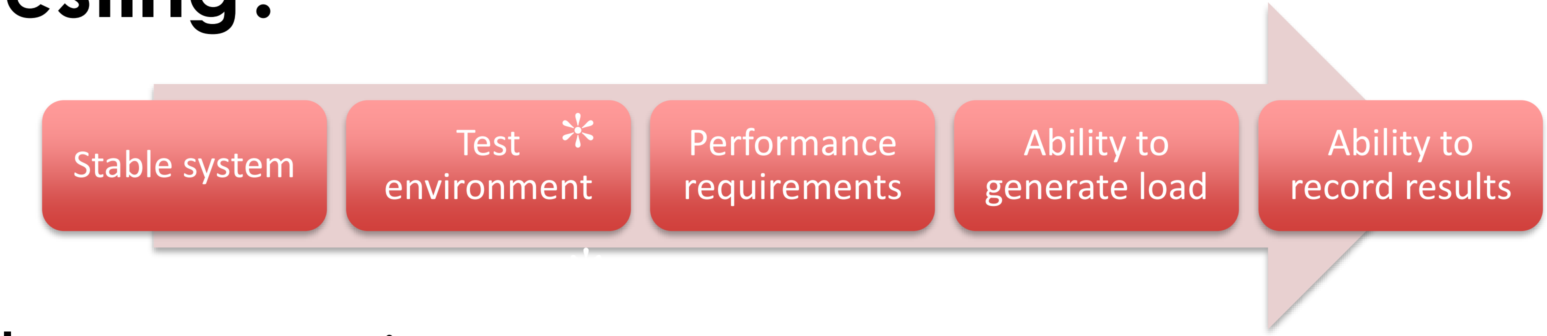
Stable system: complete or partial, but ***stable***

- Need to be able to repeat results and measure improvements

Frozen code

- Want to perform tests against a consistent code base

What’s needed for Performance Testing?



Stable system

Test environment

Performance requirements

Ability to generate load

Ability to record results

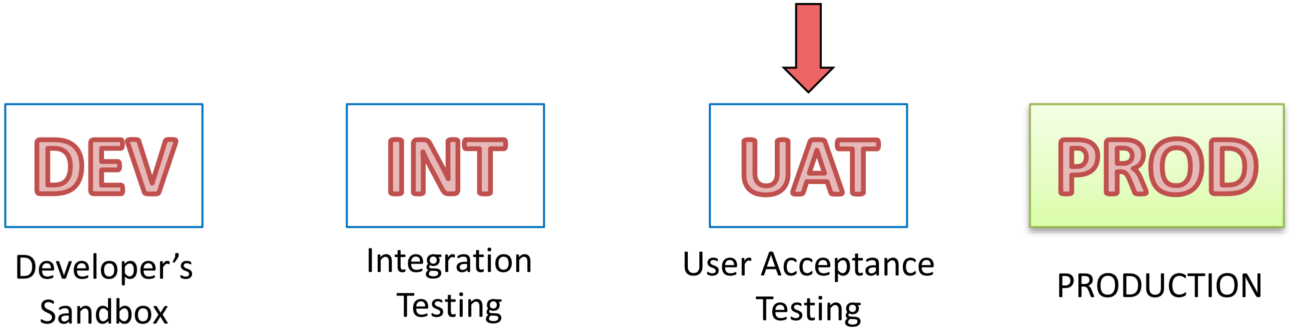
Clean test environment

Needs to be separate from other activities and other tests so you know what is being tested

Typically, UAT matches PROD

UAT used for Performance Testing

Environments:



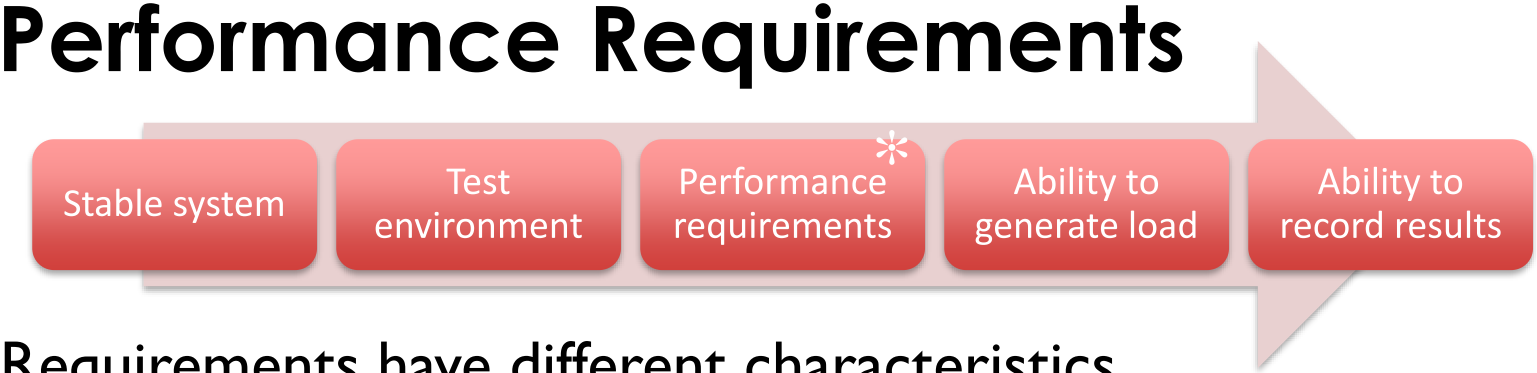
DEV
Developer’s Sandbox

INT
Integration Testing

UAT
User Acceptance Testing

PROD
PRODUCTION

Performance Requirements

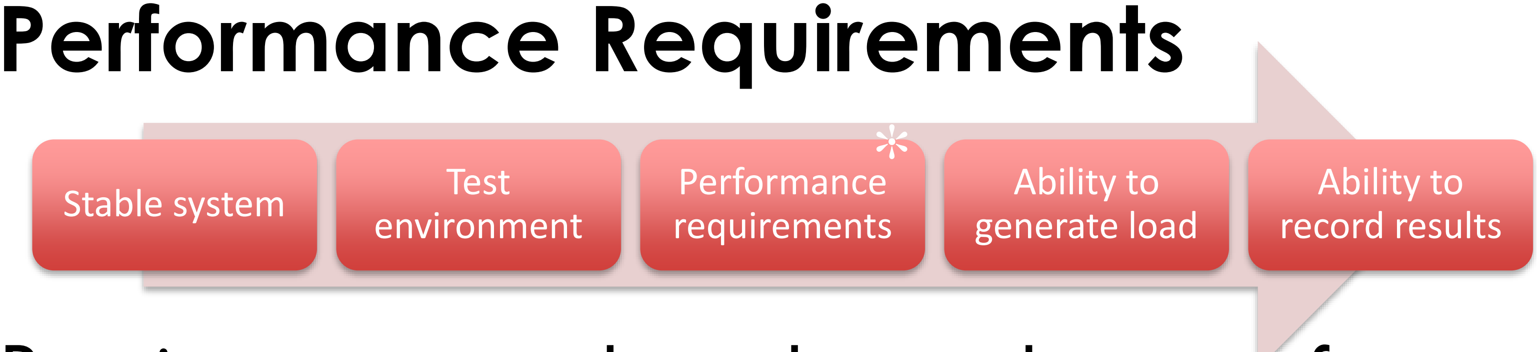


Requirements have different characteristics

- Number of simultaneous users or threads (Concurrency)
- Response Time – perceived vs (application + network)
- Throughput - volume of "transactions" over a period of time
- Scalability - ability to process increasing load of requests without a disproportionate increase in response time
  - Does the system scale linearly or exponentially?
- Reliability – Availability/uptime - system remains available regardless of load

Requirements should include the expected behavior under adverse conditions (e.g., overload)

Performance Requirements



Stable system

Test environment

Performance requirements

Ability to generate load

Ability to record results

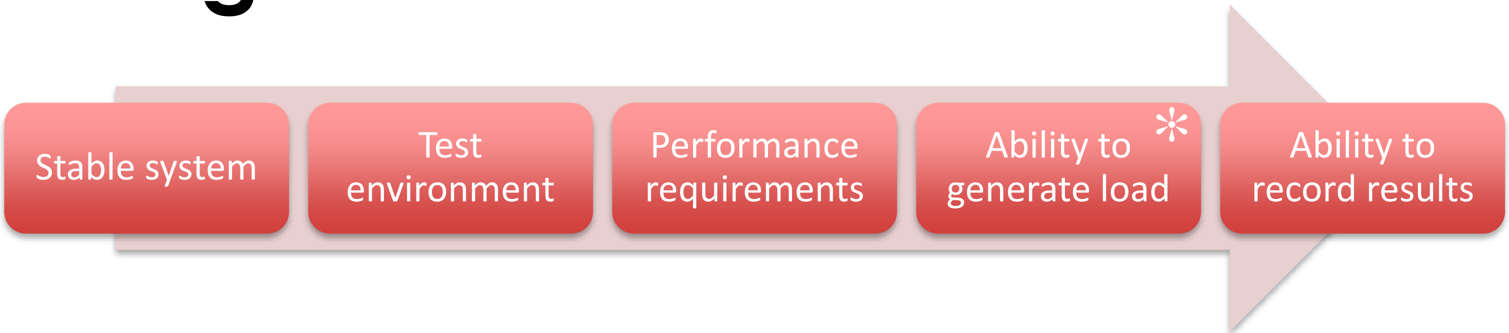
Requirements may depend upon the type of system

E.g. types of web applications:

- Publishing/subscriber sites
- Online shopping sites
- Customer self-service sites
- Trade/Auction sites

- Use requirements to script use-cases

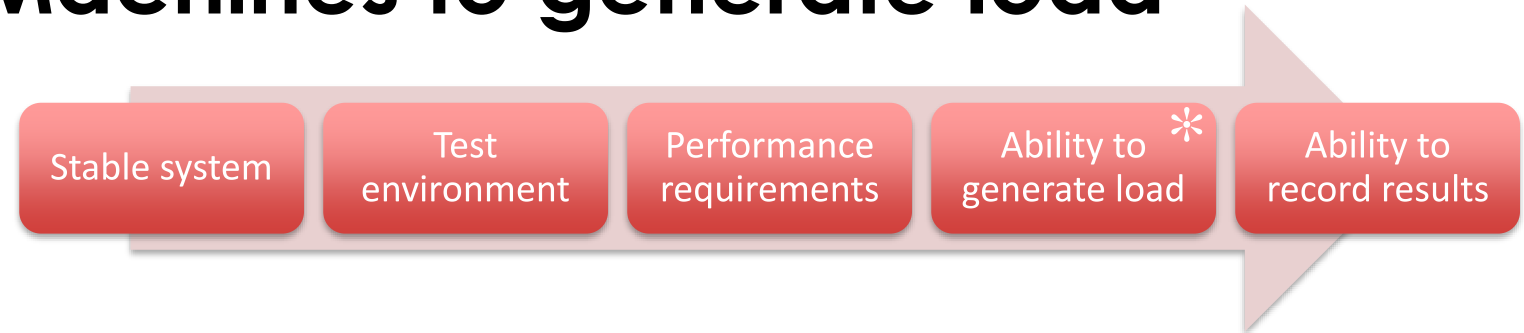
Load generator tools



- Tools to generate load – aka *Load Injectors*
- Many popular tools, both commercial and open source

http://www.softwaretestinghelp.com/performance-testing-tools-load-testing-tools/ has a list and description of capabilities

Machines to generate load



Stable system

Test environment

Performance requirements

Ability to generate load

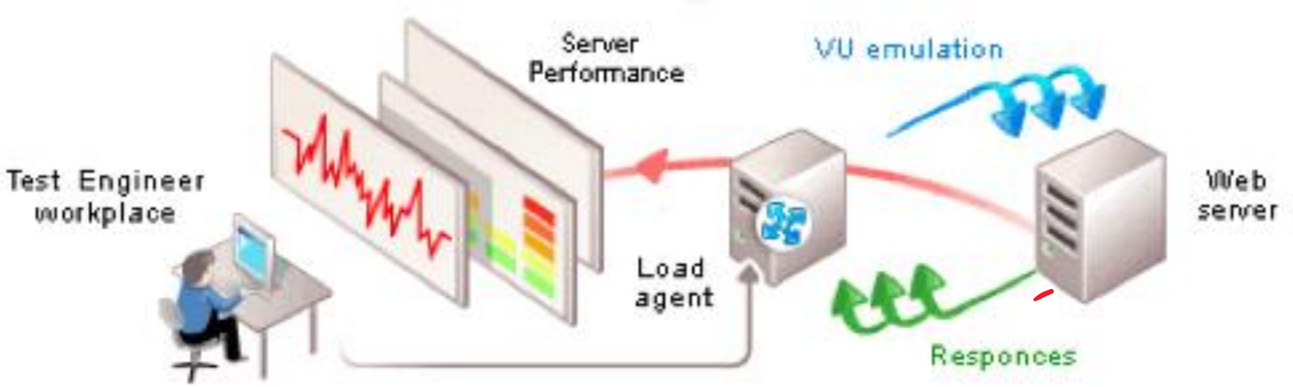
Ability to record results

Test Hardware typically required to drive the tests

- Multiple driver machines, allowing a very large number of simulated users

Test machines + load injectors == performance test environment

Load Tools For Web Clients

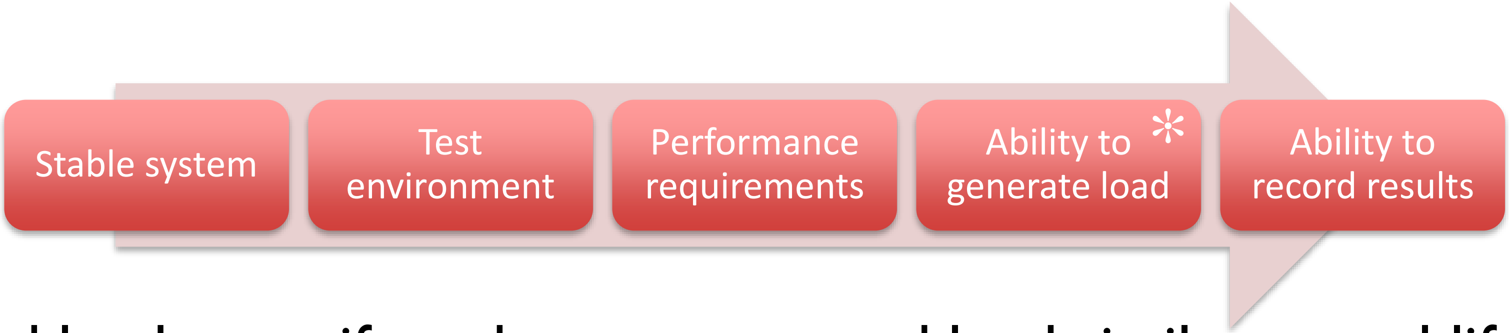


- Capture traffic at the protocol level
- Use scripting language to replay that traffic
  - Bypass the browser on playback
- Simulate many users from many machines (IP spoofing)
- Evaluating the correctness of returned page requires custom coding
- If done correctly, an application can't typically distinguish test behavior from real users

http://www.brettb.com/images/WAPT_WebApplicationLoadStressandPerformanceTesting.gif

www.PerfTestPlus.com

Workload Model



Stable system

Test environment

Performance requirements

Ability to generate load *

Ability to record results

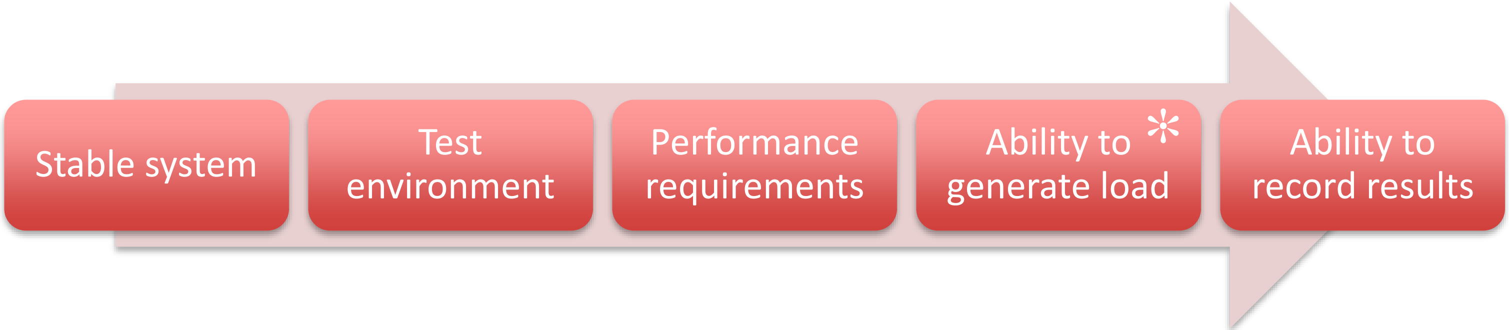
Workload - specify and generate a workload similar to real life

- Similar to operational profile or transaction mix
  - Mix of operations/transactions that represent expected system usage
- Includes
  - Volume (Number of connections or virtual users)
  - Wait Time: Think time/Idle Time between transactions
  - Other activity on server (other applications, or other system interfaces that consume resources)

Multiple workloads for a system – based upon different usage scenarios

- Holiday rush, special promotions, or potentially, or internally generated workloads such as upgrades

Workload model details



Stable system

Test environment

Performance requirements

Ability to generate load ✻

Ability to record results

A performance test simulates a particular workload

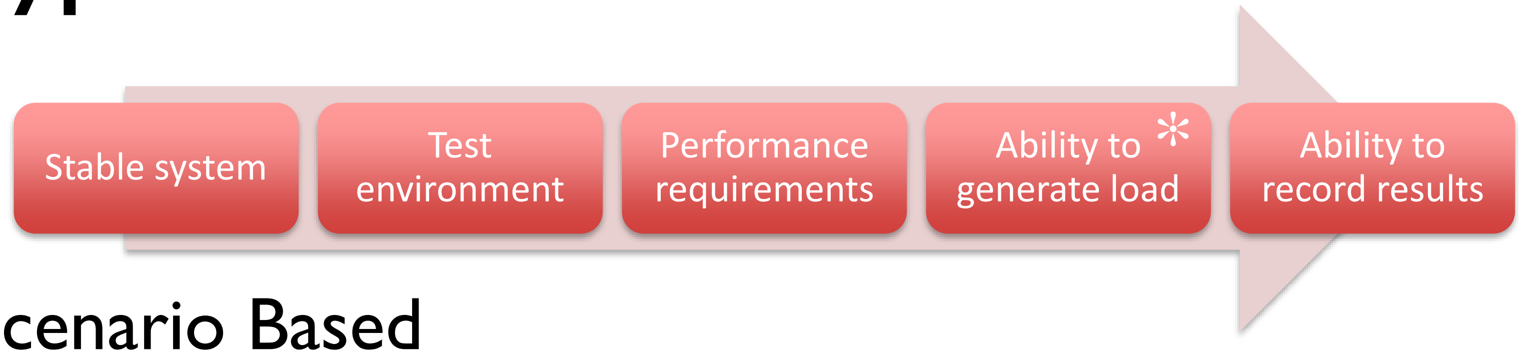
The workload is the sum of the activities placed on the system to be tested

- Workload consists of a simulated users performing a set of transactions in a specific time period

The proper workload design is critical

- If the workload is not properly designed, it is possible to reach misleading conclusions

Types of Workloads



Scenario Based

- Simulate real world conditions

Steady State

- Bring the system up to the expected load and maintain that state over time

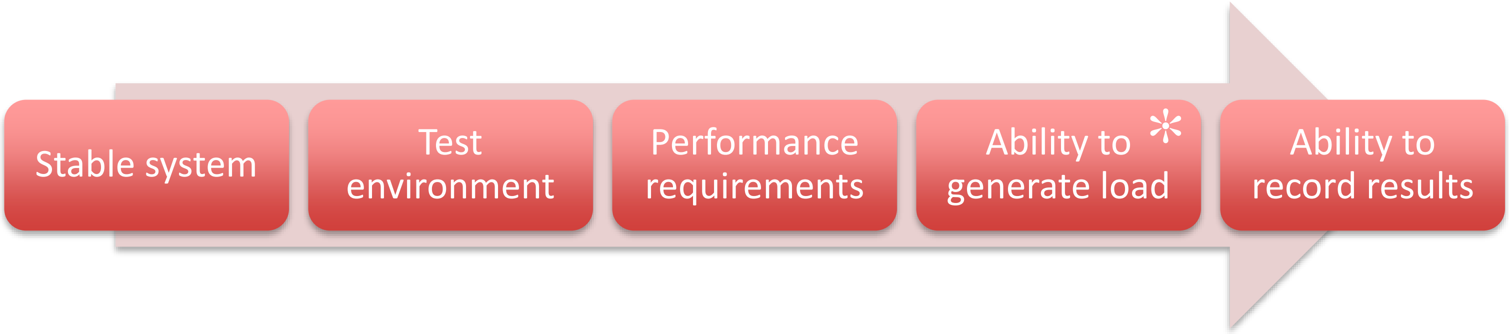
Increasing

- Start with a small load and then increase to understand performance behavior

Artificial

- Experiments to understand different loads, frequently focusing on a system component or feature

Scenario-Based Workloads

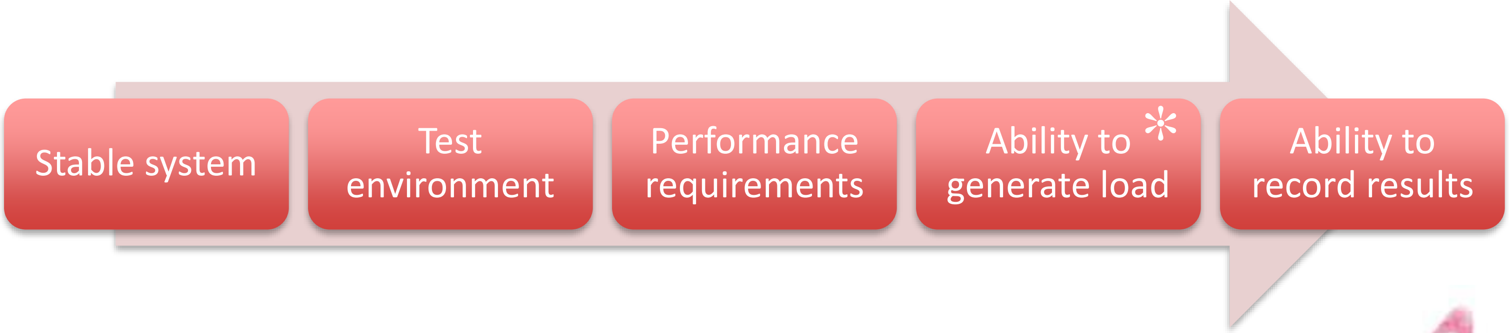


Simulate real world conditions

- Vary the number of simulated users run during a test, depending on the time of the day
    - This workload type is used for stability tests that last a long time – from a few days to a few weeks
- Vary the types of transactions based upon predefined scenarios
    - Holiday rush (Black Friday/Cyber Monday)
    - Failover

Use the operational scenarios

Steady State Workload



Determine the system's ability to run over time

Monitor resource utilization while running

- Memory
- OS resources
- …

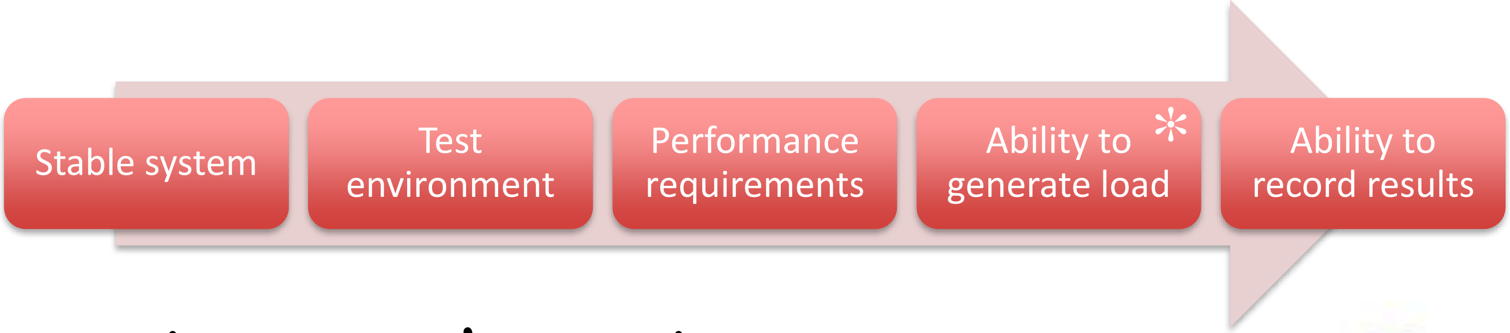
Tests the system's stability over time

Does it keep running

and running

and running?

Increasing Workload Stress Test



Stable system

Test environment

Performance requirements

Ability to generate load

Ability to record results

Determine system’s capacity

What happens when it overloads?

- Degrade gracefully?
- Crash?
- Failsafe? Fail unknown?

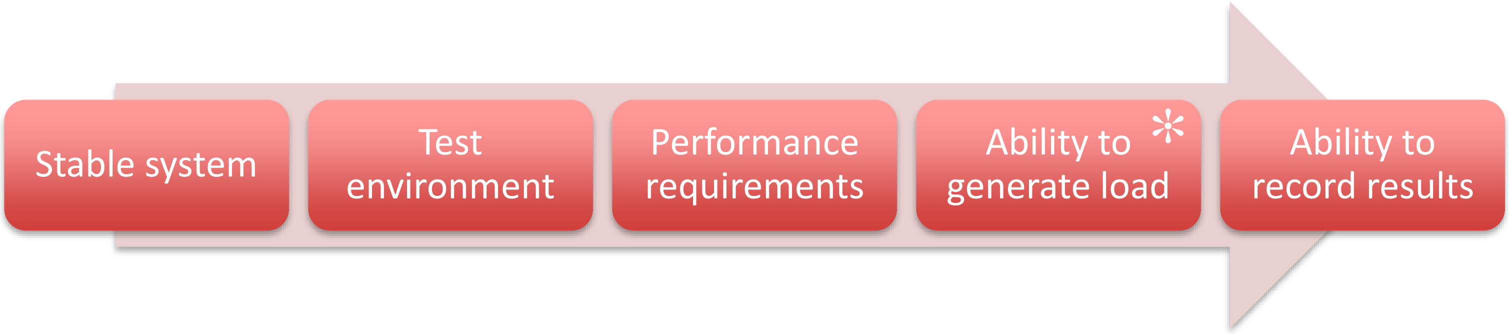
Identify the maximum number of simultaneous users the system can support

© Can Stock Photo

When does the rock roll back down?

What happens when it does?

Artificial Workloads



Stable system

Test environment

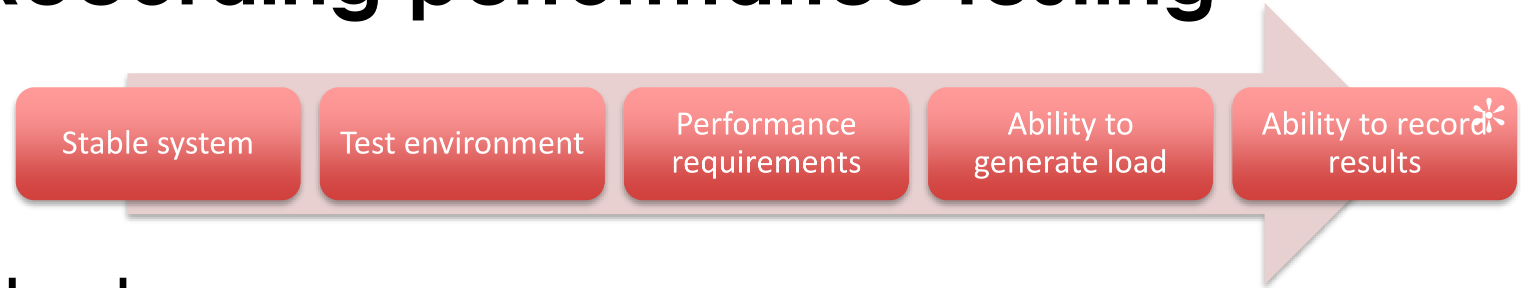
Performance requirements

Ability to generate load ✻

Ability to record results

- Typically a subset of another workload used to identify the cause of a problem
  - Simulated users are run in a way that forces the problem to occur repeatedly
  - The problem can then be isolated and diagnosed
- Used in isolation tests that are often very short, sometimes lasting only a few minutes

Recording performance testing



Stable system

Test environment

Performance requirements

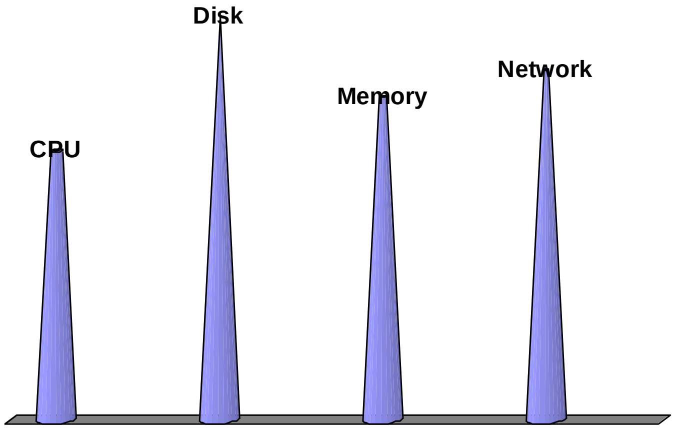
Ability to generate load

Ability to record results

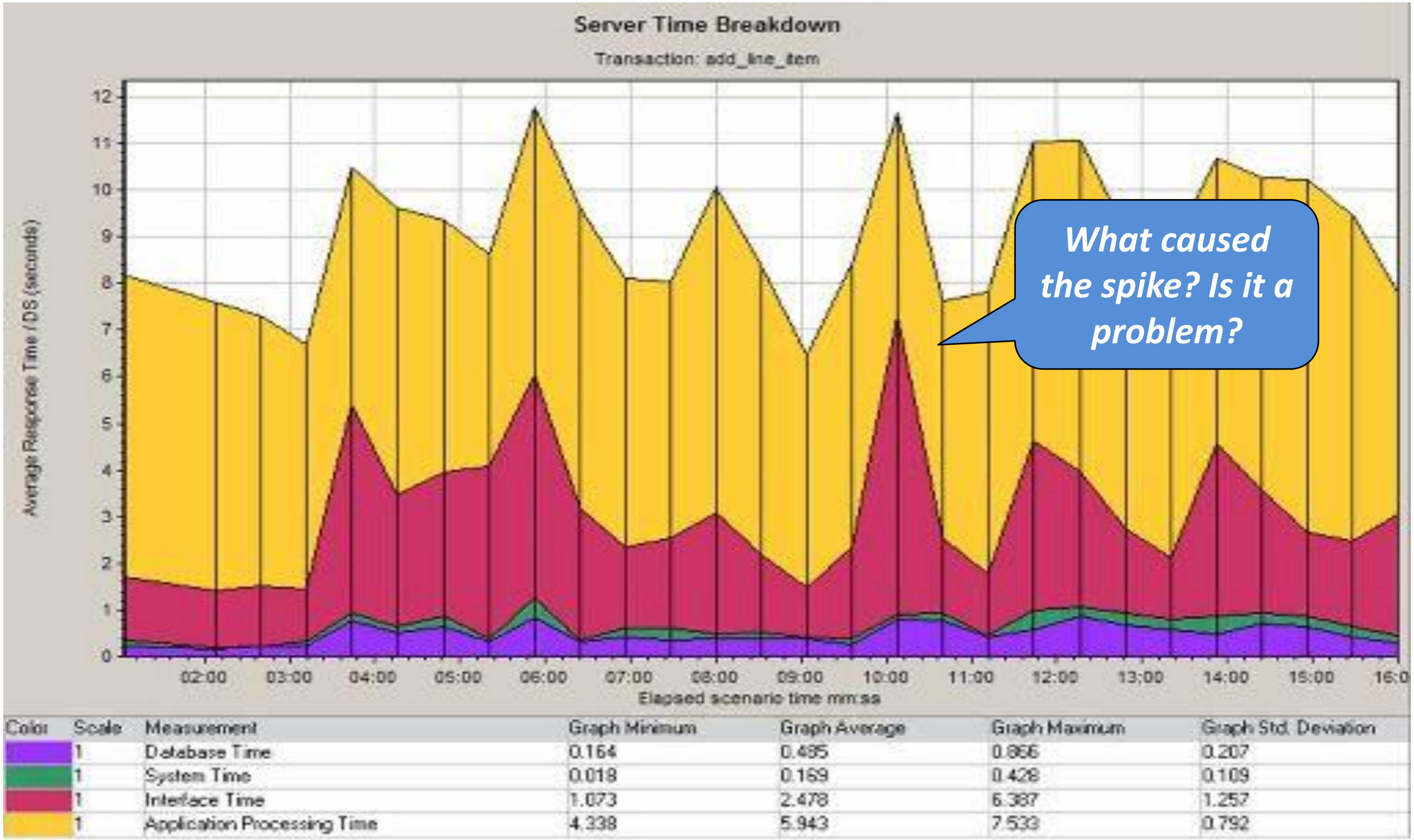
- Need to measure
  - **Resources** consumed vs. load applied
  - **Elapsed Time**
- Resources typically measured
  - Server utilization
    - CPU
    - I/O – Disk & network
    - Memory
  - Network utilization
    - Data volume, throughput, error rate

***Checkpoints** are used to "ring-fence" sections of a test to measure fine-grained response times.*

JMeter – *these are known as **Duration Assertions***



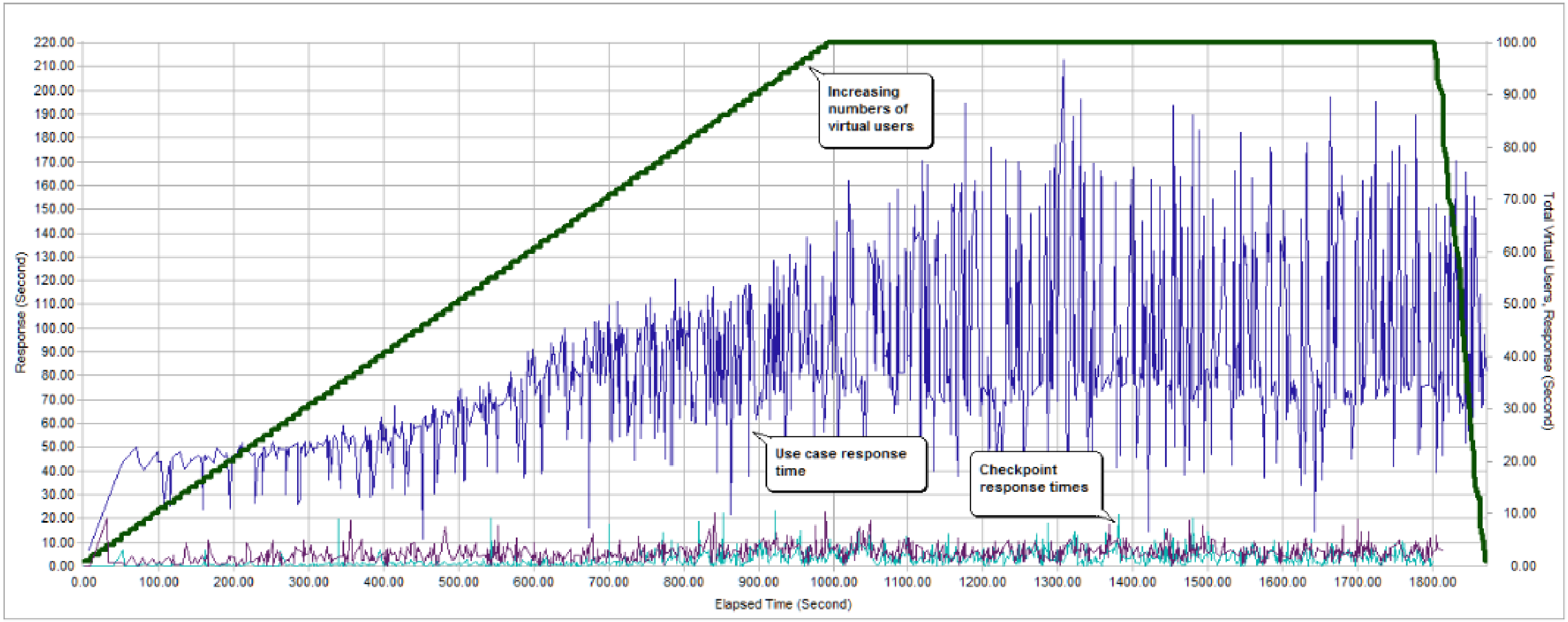
Response time - steady state test



Color	Scale	Measurement	Graph Minimum	Graph Average	Graph Maximum	Graph Std. Deviation
🟪	1	Database Time	0.164	0.485	0.866	0.207
🟩	1	System Time	0.018	0.169	0.428	0.109
🟥	1	Interface Time	1.073	2.478	6.387	1.257
🟨	1	Application Processing Time	4.338	5.943	7.533	0.792

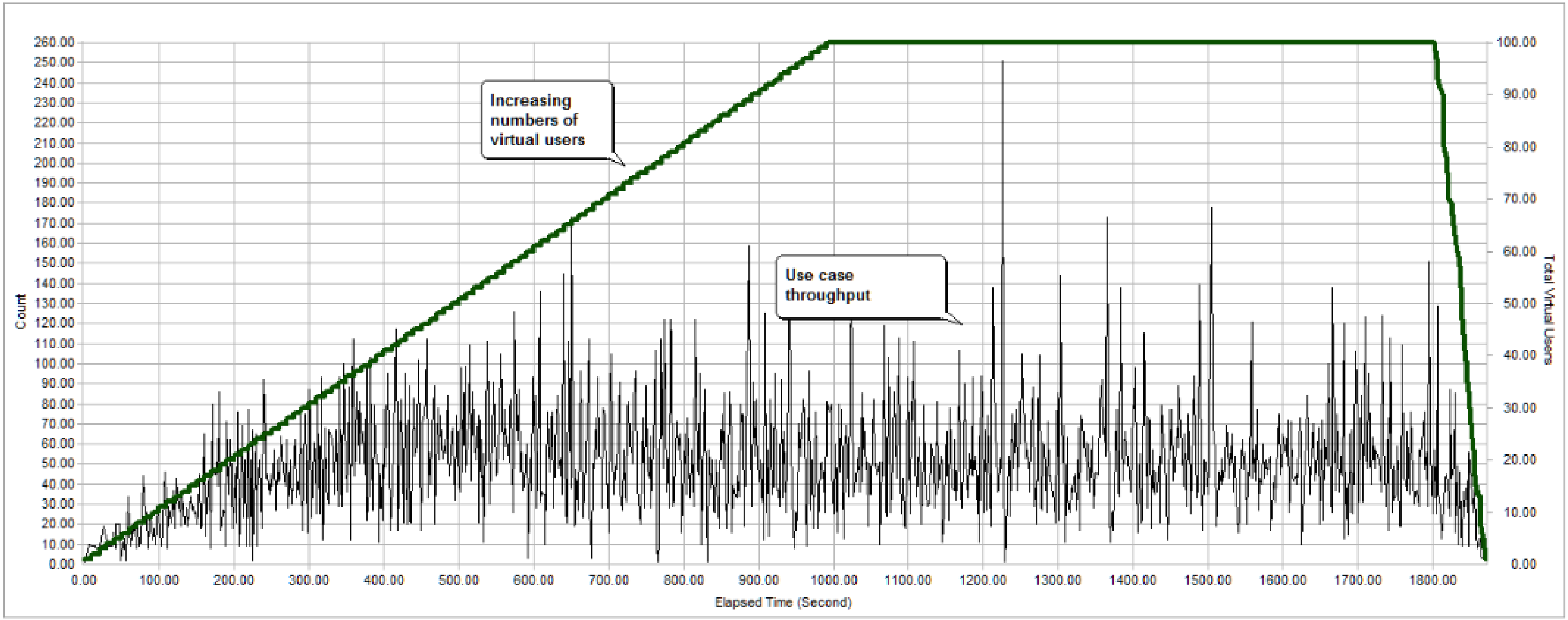
What caused the spike? Is it a problem?

Response time vs. load



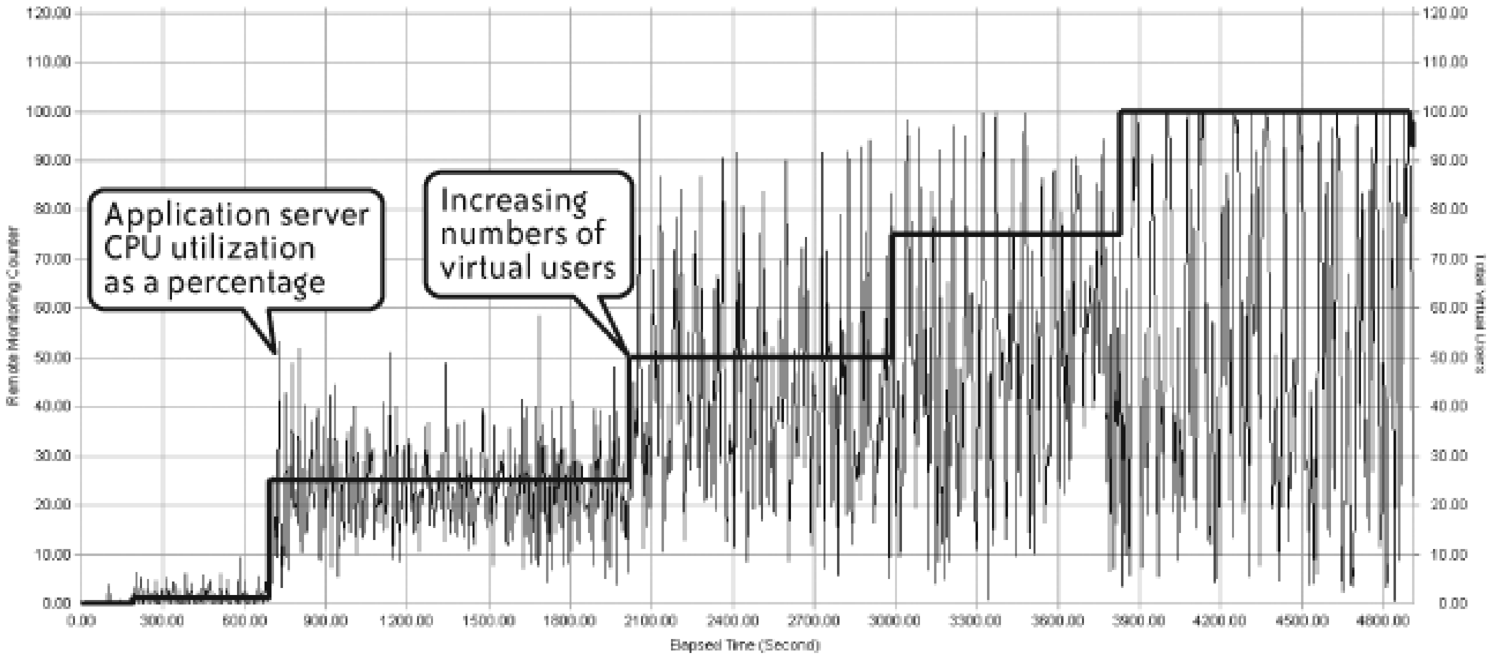
[The Art of Application Performance Testing, 2nd Edition](#), Ian Molyneaux, O’Reilly Media, 2014

Throughput vs. load



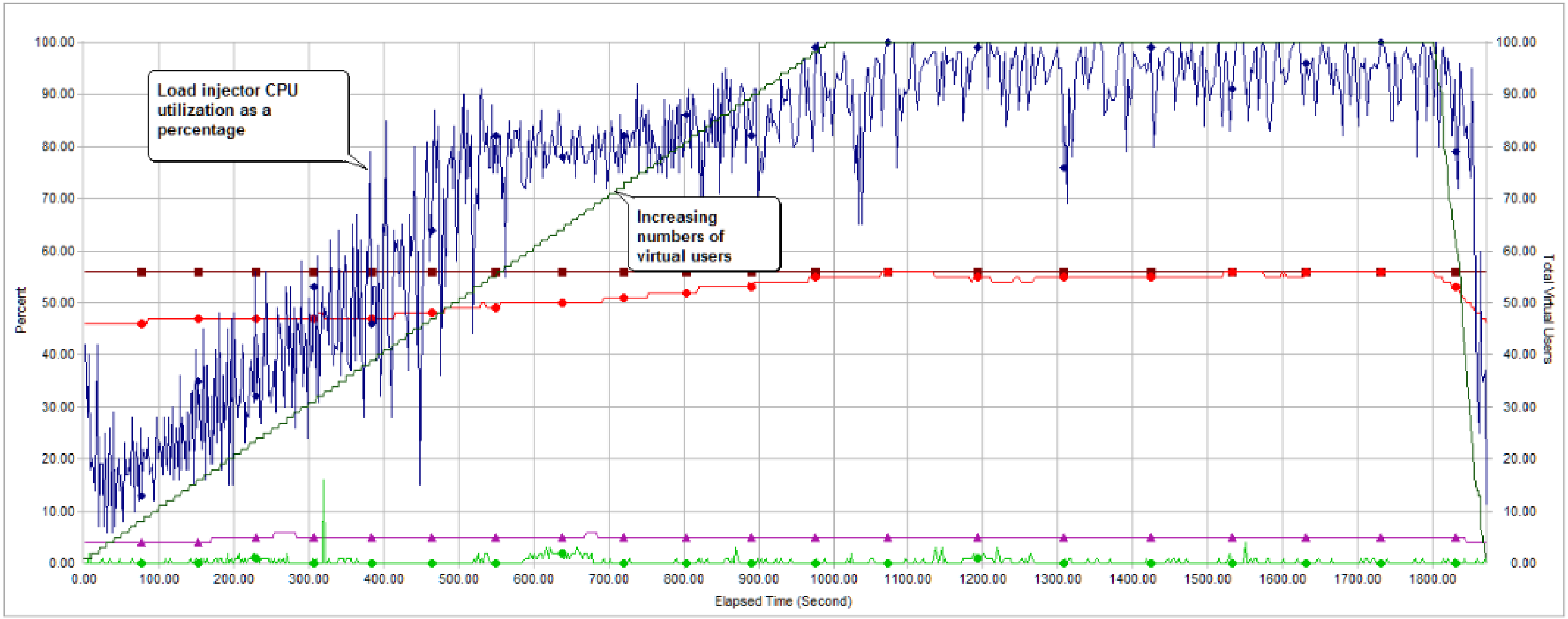
The Art of Application Performance Testing, 2nd Edition, Ian Molyneaux, O'Reilly Media, 2014

CPU vs. Load



The Art of Application Performance Testing, 2nd Edition, Ian Molyneaux, O'Reilly Media, 2014

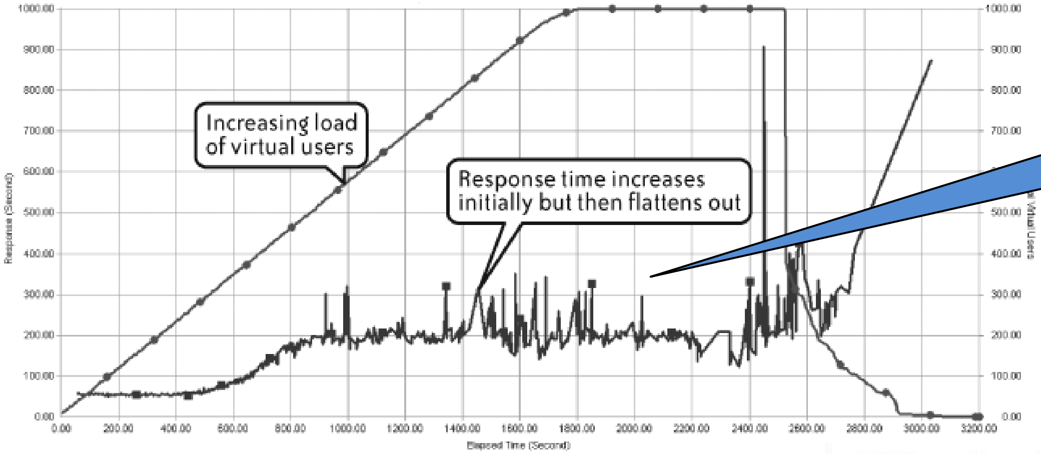
Load Injector CPU vs. Load



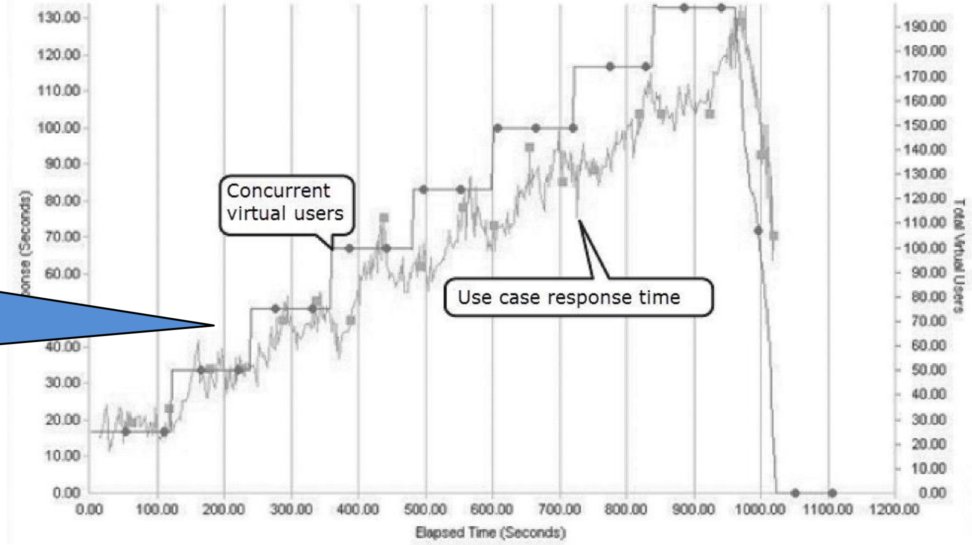
Overloaded load injectors can distort results

The Art of Application Performance Testing, 2nd Edition, Ian Molyneaux, O'Reilly Media, 2014

Response time vs. Load



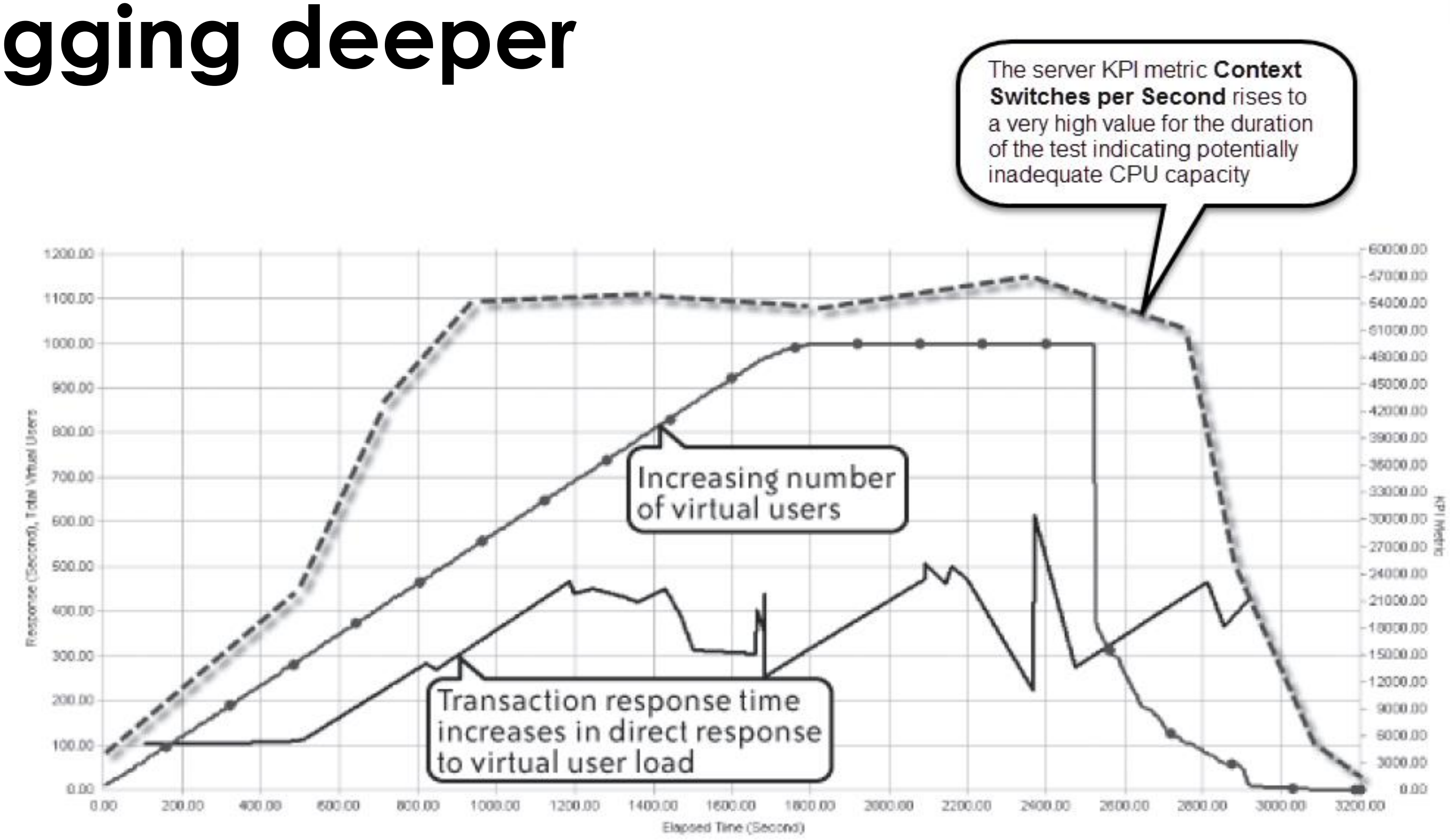
Desirable performance



Undesirable response-time behavior – response time increases lockstep with load

The Art of Application Performance Testing, 2nd Edition, Ian Molyneaux, O'Reilly Media, 2014

Digging deeper



The server KPI metric **Context Switches per Second** rises to a very high value for the duration of the test indicating potentially inadequate CPU capacity

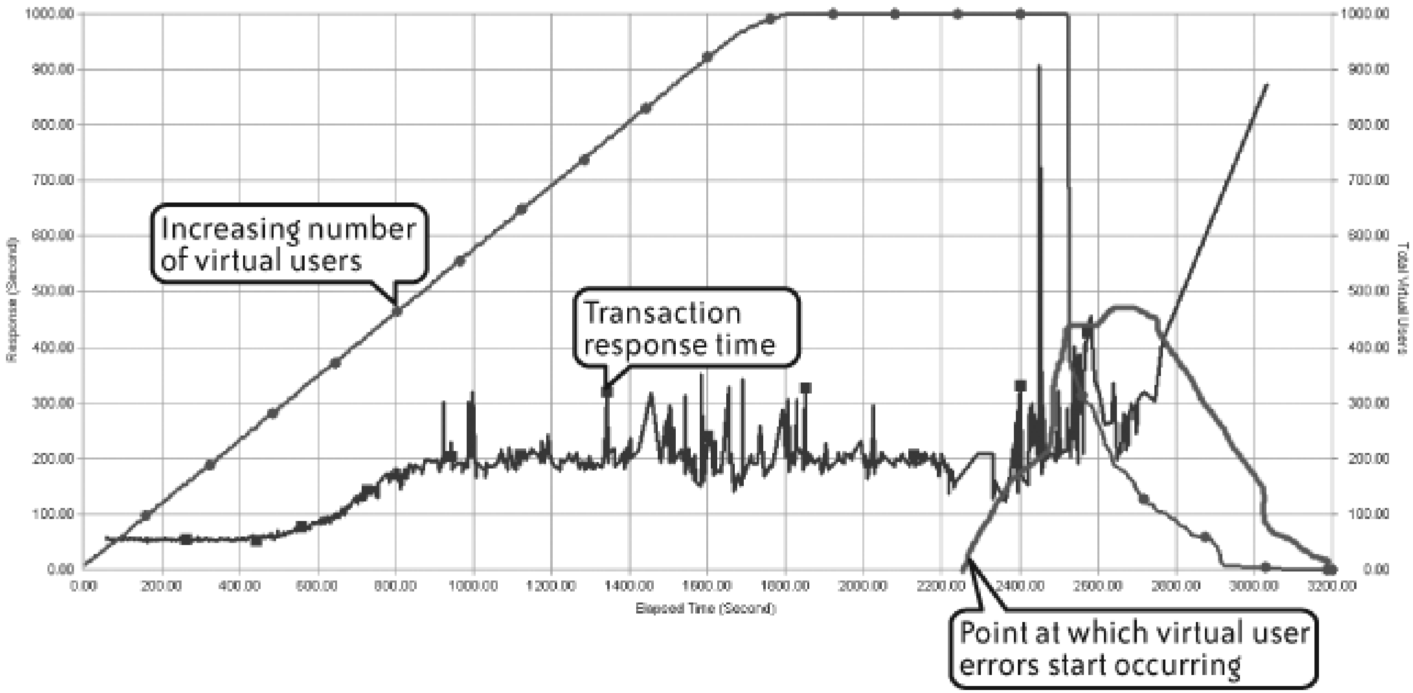
Increasing number of virtual users

Transaction response time increases in direct response to virtual user load

Superimposing "context switches/sec" metric suggests that cause of increased response time may be CPU related

The Art of Application Performance Testing, 2nd Edition, Ian Molyneaux, O'Reilly Media, 2014

Errors during the test



Increasing number of virtual users

Transaction response time

Point at which virtual user errors start occurring

Appearance of a large number of errors may indicate that some limit has been reached

The Art of Application Performance Testing, 2nd Edition, Ian Molyneaux, O’Reilly Media, 2014

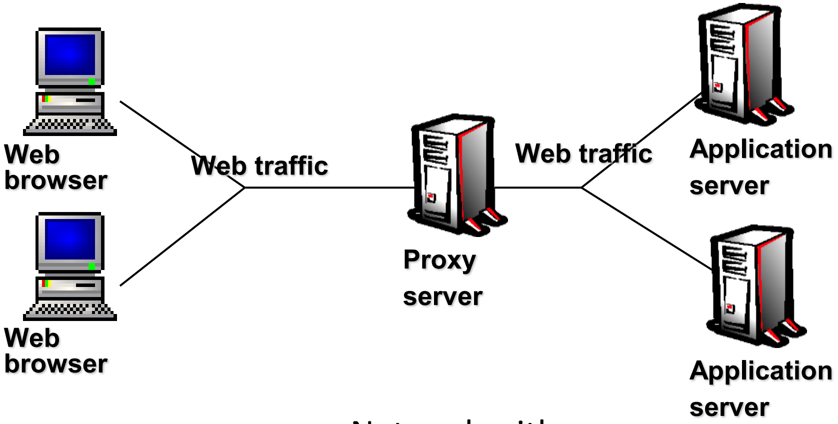
Baseline Performance

- Final result of the performance test should be baselined
- Use the baseline when monitoring app performance after deployment
- Baseline provides key metrics to help set realistic performance SLAs for the production environment

The Art of Application Performance Testing, 2nd Edition, Ian Molyneaux, O'Reilly Media, 2014

Simulating Users: Need Actual Data

Examine the traffic between actual users and the system, and use that data to create a test that mimics observed traffic



Network with proxy

Proxy Recording

For web applications, connect the web browser through an intermediary – a proxy server – to the system

The proxy captures and records all the traffic between the browser and the web server

Application Hooks

Insert hooks in the application to capture the data

Web Application Performance System Test Strategy

First - examine the entire web application using end-to-end performance testing

- Identify the general areas where bottlenecks are cropping up

Second – apply performance testing to modules with bottlenecks

- These tests focus on suspect modules within an application that are tested separately to accurately pinpoint any problems
- These tests are also called isolation tests
- Isolation tests help testers to identify specific problems

Performance Testing Summary

- Performance tests must be carefully designed and planned around the specific business goals and objectives
- Testers must be provided with a quantified set of requirements for the performance of the system. Each important performance requirement must be tested, e.g.,
  - Response Time
  - Throughput
  - Latency
  - Concurrency
- Good Performance Engineering is a valuable skill

Reliability Testing

Good reference: Concise Guide to Formal Methods – Regan 2017

Software Reliability

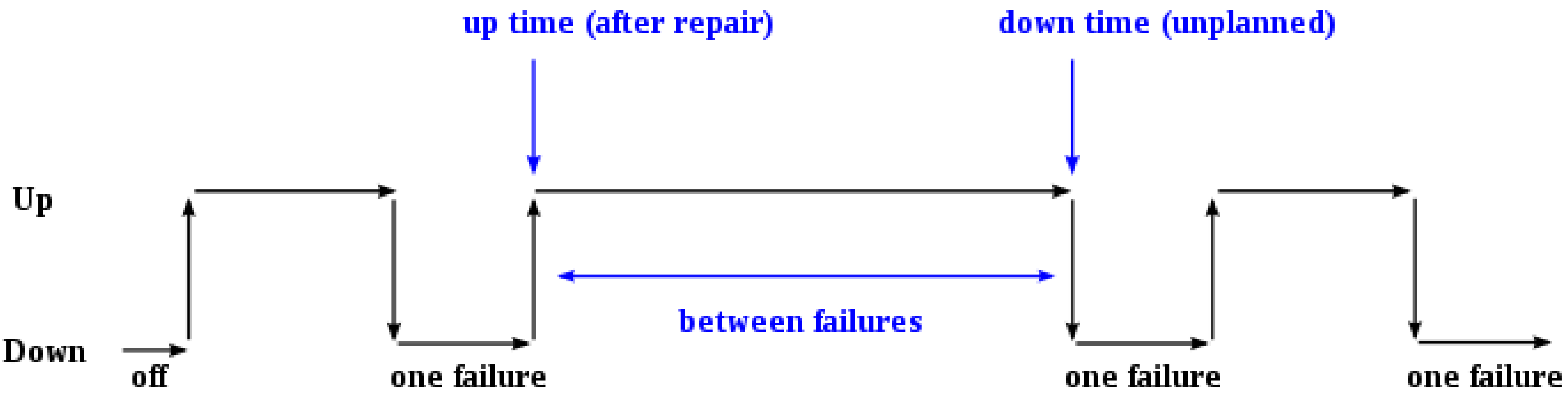
Software Reliability is the ***probability*** that the software system will function ***without failure*** under a given ***environment*** and during a specified ***period of time***

- The estimate is **probabilistic**, not absolute
- ***Without failure*** implies that we must define "failures"
  - Is a web site being not being available a failure?
  - Is response time > 3 seconds a failure?
- ***Under a given environment*** defines where the reliability can be verified
- **During a specified period of time**
  - Reliability of 99.999% for ***two seconds*** vs
  - Reliability of 99.999% for ***two years***

Software Reliability

Software Reliability sometimes expressed as the MTBF

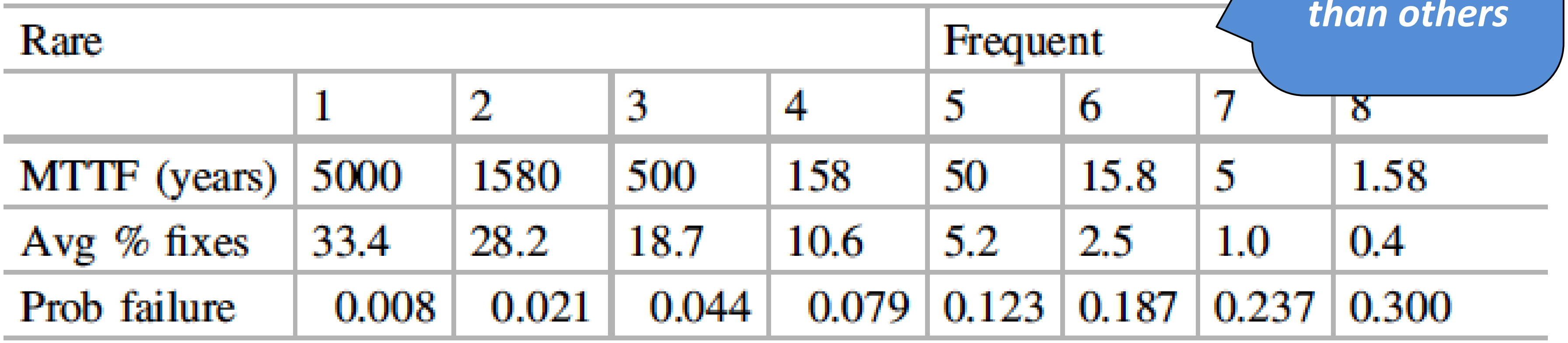
Mean Time Between Failures



Time Between Failures = { down time - up time}

Software Failure Characteristics

Some bugs are more important than others



Rare					Frequent			
	1	2	3	4	5	6	7	8
MTTF (years)	5000	1580	500	158	50	15.8	5	1.58
Avg % fixes	33.4	28.2	18.7	10.6	5.2	2.5	1.0	0.4
Prob failure	0.008	0.021	0.044	0.079	0.123	0.187	0.237	0.300

One-third of the errors have MTTF of 5000 years!

The four rightmost columns include ~10% of the faults
- ~100 times more likely to occur than first column

The two rightmost columns include ~2% of the faults
- ~1000 times more likely to occur than first two columns

Need to identify the defects mostly likely to occur

Adam’s study of 9 Large IBM Projects - 1984

Conclusion from this case study

Coverage testing is not as effective in improving reliability

vs.

Usage testing, which would allocate most testing to fix issues that would occur more than 50 % of the time

Cleanroom Methodology

Conceived at IBM to improve reliability

Employs *statistical usage testing vs. coverage testing*

Coverage testing – design tests to cover every possible path through the program

Statistical usage testing – executes tests based on probability of expected use

How? Operational Profile Testing

The reliability of a software product depends on the environment in which it executes and how it is used

We need to test software as it would be used in the field to estimate the reliability of software

→ Operational Profile Testing

Identify the ***operations*** and the ***frequency*** of those operations

Operational Profile Testing

The *Operational Profile* is

A ***quantitative*** characterization of how the software will be used in the field

Consists of a complete set of operations with their probabilities of occurrence

- Probability of occurrence refers to invocations of all operations
- Example: if 17 out of every 100 invocations of a system is "purchase tickets" then the probability of occurrence = 0.17

A sample operational profile for an ATM machine

Use Case	Probability of Occurrence
Cash Withdrawal	0.53
Checking Deposit	0.15
Savings Deposit	0.14
Funds Transfer	0.08
Balance Inquiry	0.06
Restock	0.02
Collect Deposits	0.02
Total	**1.00**

Developing an Operational Profile

Often done by systems engineers/marketing and product personnel, but testers should be involved too

Five principle steps in developing an operational profile:

1. Identify initiators of operations - actors
2. Create operations list – scenarios or transactions
3. Review operations list
4. Determine occurrence rates
5. Determine occurrence probabilities

Start developing the profile in the requirements phase and refine iteratively in later phases

ATM Actors -- Initiators

Use Case	Actors
Cash Withdrawal	Bank Customer, Crook, Customer Bank
ATM Cash Restocking	Operator & Guard, Bank Manager

Creating an operations list

- Use scenarios from Use Cases to identify specific operations
- For each initiator, list
  - Rough guideline: each operation should have more than 100 deliverable source lines different from another operation.
  - High probability that an operation will contain a fault not in any other operation
- Sources:
  - System requirements
  - Work process flow diagrams
  - User manuals, prototypes, and information on previous releases
- Include "housekeeping operations" that (re)initialize or clean up data
- Should execute each operation at least once in test (unless it has a very low occurrence probability and is non-critical)

ATM Possible Scenarios?

Use Case	Actor	Scenario
Cash Withdrawal	Bank Customer	
Cash Withdrawal	Bank Customer	
Cash Withdrawal	Crook	
ATM Cash Restocking	Operator & Guard	
ATM Cash Restocking	Operator & Guard	
ATM Cash Restocking	Bank Manager	

Think about different scenarios which may be driven by different types of input variables

ATM Scenarios

Use Case	Actor	Scenario
Cash Withdrawal	Bank Customer	Wrong PIN entered once, request $75
Cash Withdrawal	Bank Customer	PIN OK, deposit $300, request $50
Cash Withdrawal	Crook	Stolen card inserted, valid PIN entered
ATM Cash Restocking	Operator & Guard	ATM opened, cash dispenser empty, $15,000 is added
ATM Cash Restocking	Operator & Guard	ATM opened, cash dispenser is full
ATM Cash Restocking	Bank Manager	ATM opened, cash dispenser and deposits available

Determining Operation Rates

Operation Rate = Number of occurrences of operation over some time period

- E.g. : 10 deposits per hour (per ATM)

Strategies:

- Look for
  - Existing field data
  - System logs
  - Existing business data; the original business case for the product
- Ask a marketing person
- Record field operations
- Instrument code
- Estimate!
  - Apply the Delphi method with multiple experts.
  - Group the low probability operations and assign equal probability to each
  - Nothing to go on? Assign all operations equal probability

ATM Scenarios

Use Case	Actor	Operations
Cash Withdrawal	Bank Customer	1) Enter PIN – OK 2) Enter PIN – Wrong 3) Enter PIN – Wrong > 2 Times
Cash Withdrawal	Bank Customer	4) Deposit Check 5) Withdraw Cash
Cash Withdrawal	Crook	6) Stolen card inserted
ATM Cash Restocking	Operator & Guard	7) ATM opened 8) Restocked 9) Deposits Removed
ATM Cash Restocking	Operator & Guard	10) ATM full – not restocked
ATM Cash Restocking	Bank Manager	11) ATM opened 12) Cash rebalanced 13) Deposits Removed

Determine Operation Probabilities

Need to measure/estimate *how often* each operation executes

- Relative Frequency

How long each one takes (duration) to determine the probability of execution

ATM Scenarios

Use Case	Actor	Operations	Relative Freq	Dur	Dur * Freq	Oper. Prob
Cash Withdrawal	Bank Customer	1) Enter PIN – OK 2) Enter PIN – Wrong 3) Enter PIN – Wrong > 2 Times	100 50 10	10 20 20	1000 1000 200	26.99% 26.99% 5.40%
Cash Withdrawal	Bank Customer	4) Deposit Check 5) Withdraw Cash	50 100	10 10	500 1000	13.49% 26.99%
Cash Withdrawal	Crook	6) Stolen card inserted	0.5	10	5	0.13%
ATM Cash Restocking	Operator & Guard	7)ATM opened 8) Restocked 9) Deposits Removed	0.01 0.01 0.01	15 10 10	0.15 0.10 0.10	0.00% 0.00% 0.00%
ATM Cash Restocking	Operator & Guard	10) ATM full – not restocked	0	10	0.01	0.00%
ATM Cash Restocking	Crook	11) Cash in Dispenser Removed 12) Alarm Triggered 13) Deposits Removed	0 0 0	15 10 10	0.00015 0.0001 0.0001	0.00% 0.00% 0.00%

Using operational profile for testing

Allocate testing based upon operational probabilities to drive highest *reliability for the aggregate*

% of testing == % of occurrence

Something we didn't cover but that is also important, is that we need to also consider the ***criticality of operations*** when testing!!

E.g. Nuclear plant shutdown is maybe a low-occurring operation, but has high criticality

Predicting Reliability

Given the values

Use Case	Probability of Execution	Predicted Failure Rate
Cash Withdrawal	0.53	0.001
Checking Deposit	0.15	0.002
Savings Deposit	0.14	0.002
Funds Transfer	0.08	0.003
Balance Inquiry	0.06	0.003
Restock	0.02	0.003
Collect Deposits	0.02	0.003
Total	1.00	

Predicting Reliability

Use Case	Probability of Execution	Predicted Failure Rate	Contribution to System Failure Rate
Cash Withdrawal	0.53	0.001	0.0005
Checking Deposit	0.15	0.002	0.0003
Savings Deposit	0.14	0.002	0.0003
Funds Transfer	0.08	0.003	0.0002
Balance Inquiry	0.06	0.003	0.0002
Restock	0.02	0.003	0.0001
Collect Deposits	0.02	0.003	0.0001
Total	1.00		0.0017

Reliability Testing

- Want to find defects that have greatest impact on product’s reliability
  - Use the operational profile to guide testing (and bug fixing) to have the biggest impact on reliability
- Combine reliability testing with performance/stress testing by using the operational profile
  - Should use the operational profile (or something similar) to predict performance
  - Use for stress testing to simulate actual operations

Summary: Operational Profiles

Operational Profiles are an important way to do reliability estimation

Method:

- Create an operational profile which models actual system usage
- Determine the failure rates of the individual transactions
- Calculate expected system reliability
- Use the expected usage volume in operation to calculate actual reliability after deployment

Usage-based testing vs. coverage-based testing → factor of 21 improvement in reliability. Musa -- 1994

Questions?