LOG 8371E
Software Quality Engineering

Lecture 05:
Software Performance Engineering

Armstrong Foundjem Ph.D. — Winter 2024

Obamacase website crashed on the day of launch

The Failed Launch Of www.HealthCare.gov

By **ABC123**

Alumni

MODIFIED NOV 18, 2016

Next:

ImaginBank and its war against fintech

The US Government's failed launch of the Healthcare.gov website highlights issues with integrating technology into a large bureaucratic organization.

"I'm going to try and download every movie ever made, and you're going to try to sign up for Obamacare, and we'll see which happens first" – Jon Stewart challenging Kathleen Sebelius (former Secretary of Health and Human Services) to a race.

(Ultra) Large-Scale Software Systems

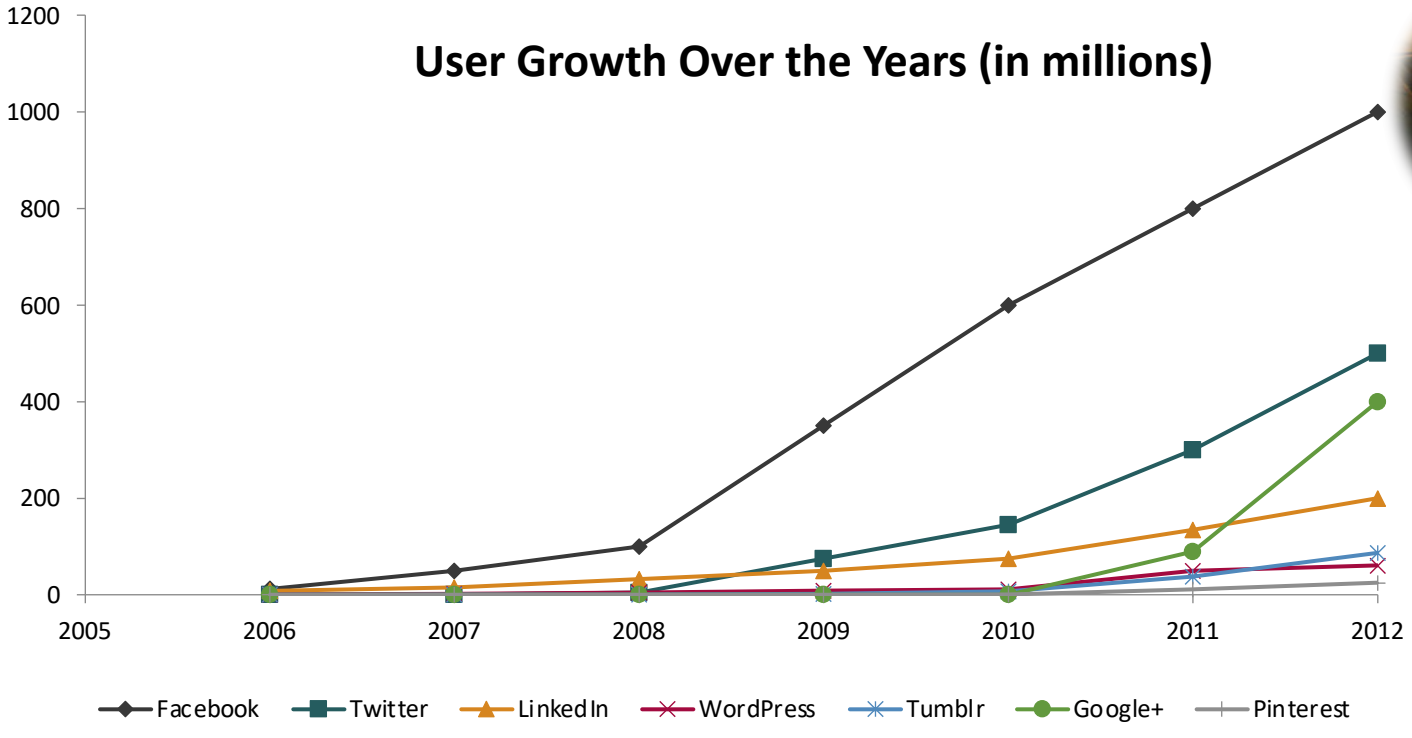
4 million users

2600-3000 req/sec on most weekdays

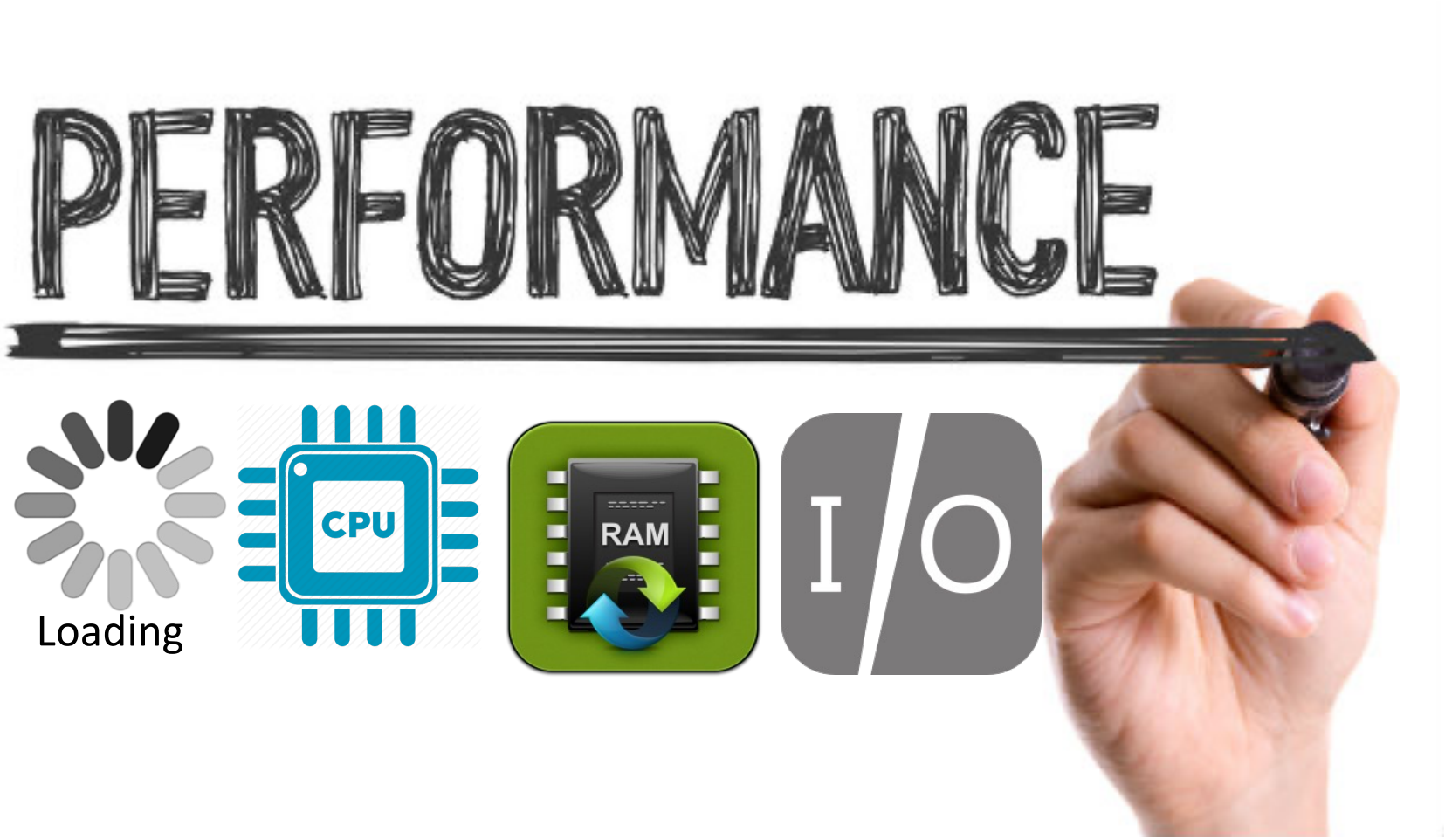
450 million active users

> 50 billion messages every day

Rapid Growth and Varying Usage Patterns



Software system failures are often due to performance issues rather than functional bugs



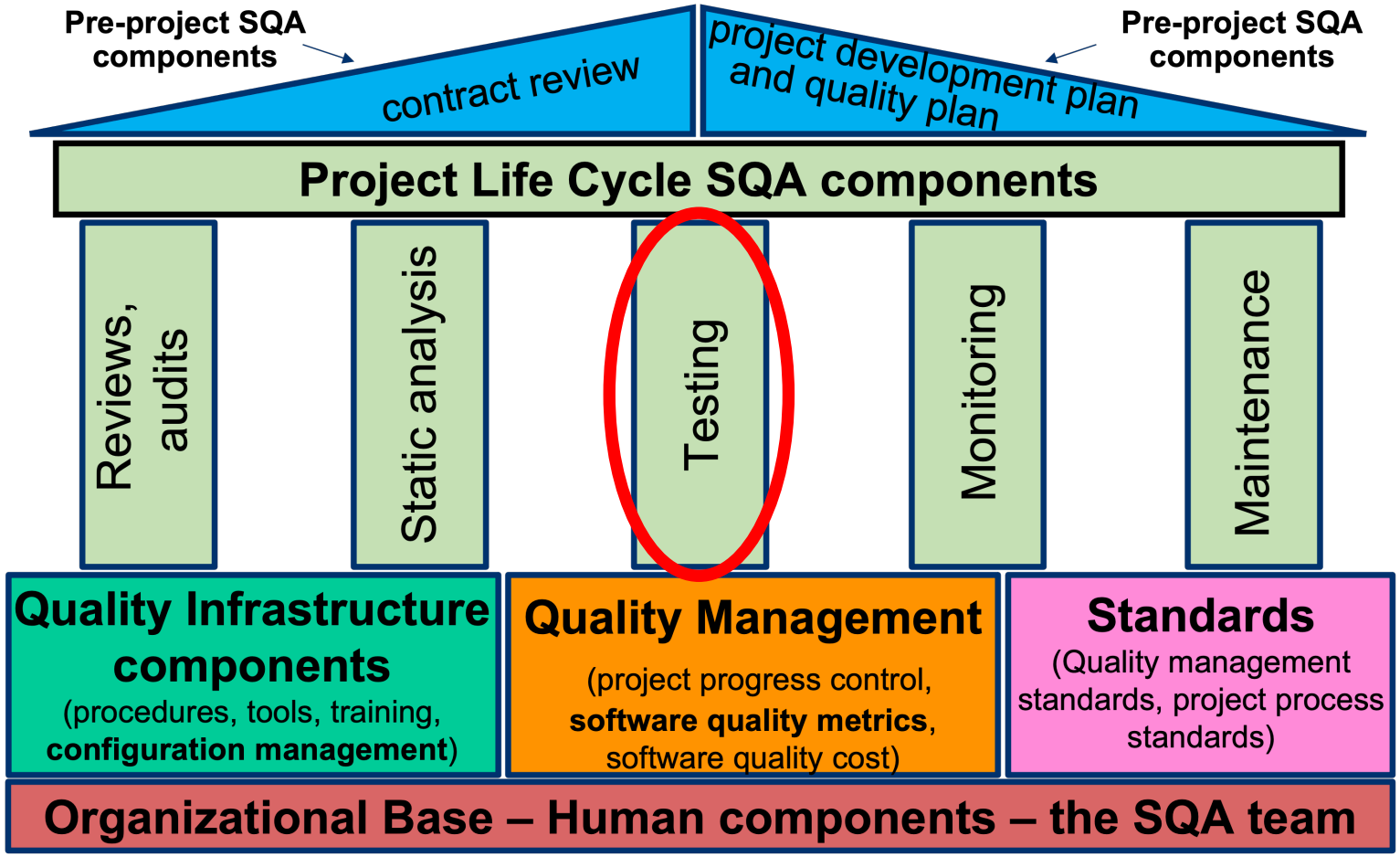
Software system failures are often due to performance issues rather than functional bugs

One hour global outage
lost $7.2 million in revenue
(02/24/09)

Flickr outage impacted
89 million users
(05/24/13)

A page load slowdown of only one second could cost $1.6 billion

Today's topic in the SQA system



Agenda

- Software Performance Engineering (SPE)
- Performance testing
- Profiling

- Resources:
  - Trevor Warren, Body of Knowledge on Systems Performance Engineering. https://tangowhisky37.github.io/PracticalPerformanceAnalyst/about/
  - Gregg, Brendan. *Systems performance: enterprise and the cloud*. Pearson Education, 2013.
  - Jain, Raj. *The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling*. Wiley professional computing, 1991.

Software Performance Engineering (SPE)

Performance Efficiency (ISO 25010)

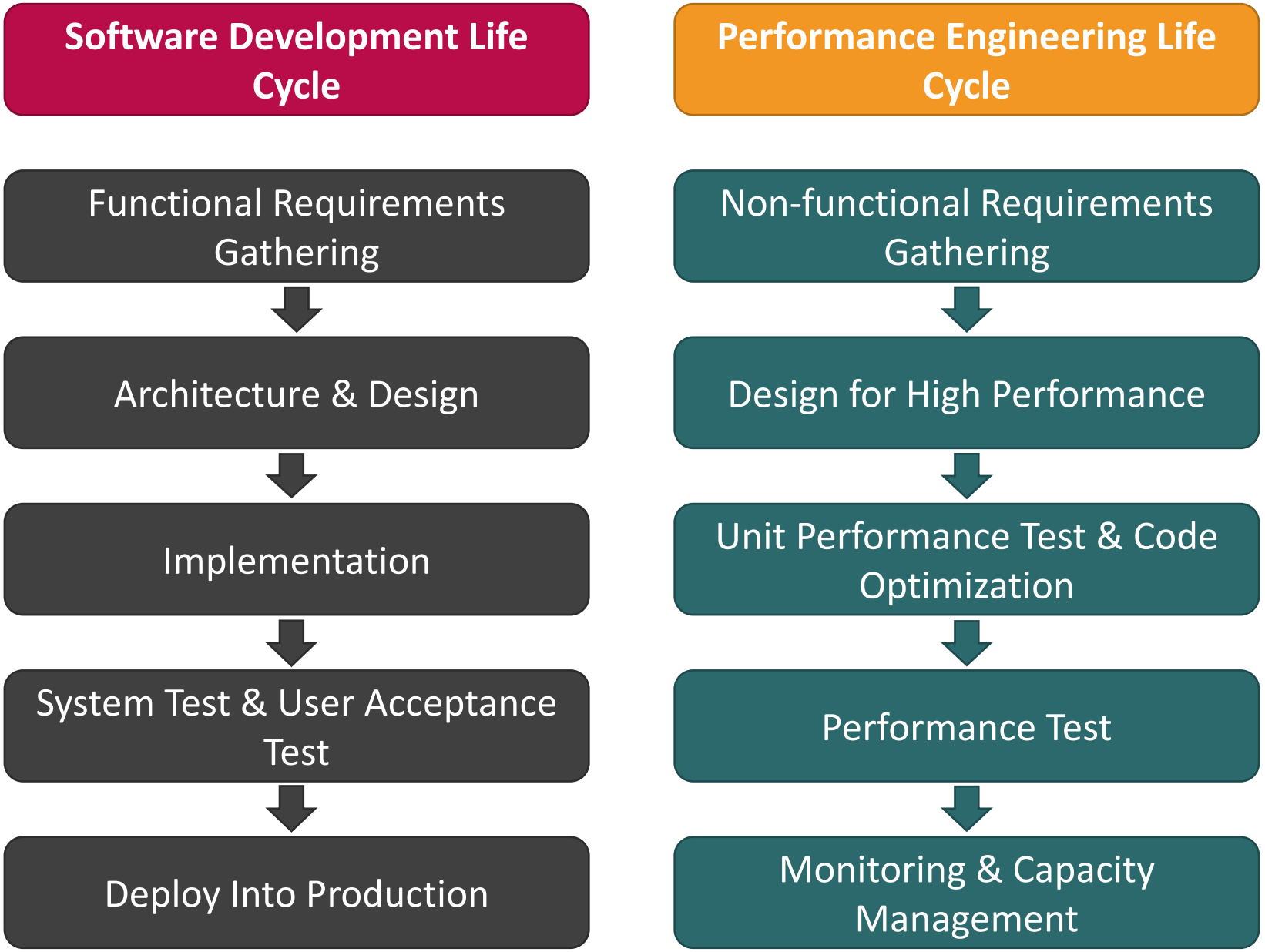
Quality (sub) factor	Description
Performance efficiency	Performance relative to the amount of resources used under stated conditions
• **Time behavior**	Degree to which the response and processing times and throughput rates of a product or system, when performing its functions, meet requirements
• **Resource utilization**	Degree to which the amounts and types of resources used by a product or system when performing its functions meet requirements
• **Capacity**	Degree to which the maximum limits of a product or system parameter meet requirements

Software performance

- Performance measures the efficiency of the software against the constraints of time and resource allocation.
- There are several indicators to capture and evaluate performance.
  - ✓**Response Time:** The total elapsed time between submission of a request and receipt of the response.
  - ✓**Processing Rate/Throughput:** The total completions per unit time, e.g. Transactions/Sec.
  - ✓**Utilization:** The ratio of busy time to total time (how busy or free the resources within a given system are).
  - ✓Other indicators (e.g., capacity, battery/power consumption)
- It is possible to consider the performance of an entire system (including hardware and software) or part of the system such as a software component.

Software Performance Engineering (SPE)

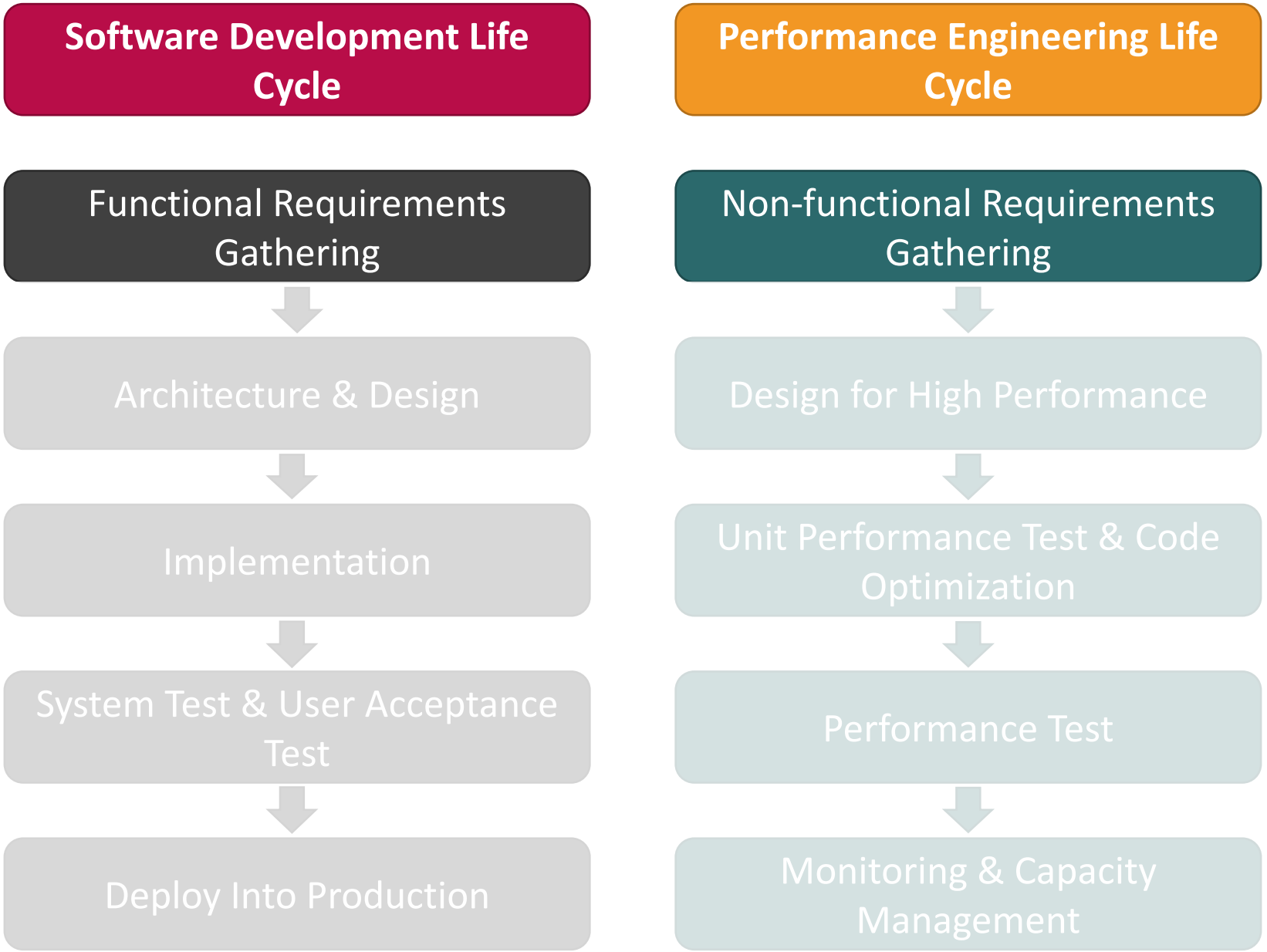
- The set of tasks or activities that need to be performed across the Software Development Life Cycle (SDLC) to meet the documented Non-Functional Requirements (Performance, Scalability, Availability, Reliability, etc.)



Source: https://tangowhisky37.github.io/PracticalPerformanceAnalyst/pages/spe_fundamentals/performance_engineering_101/

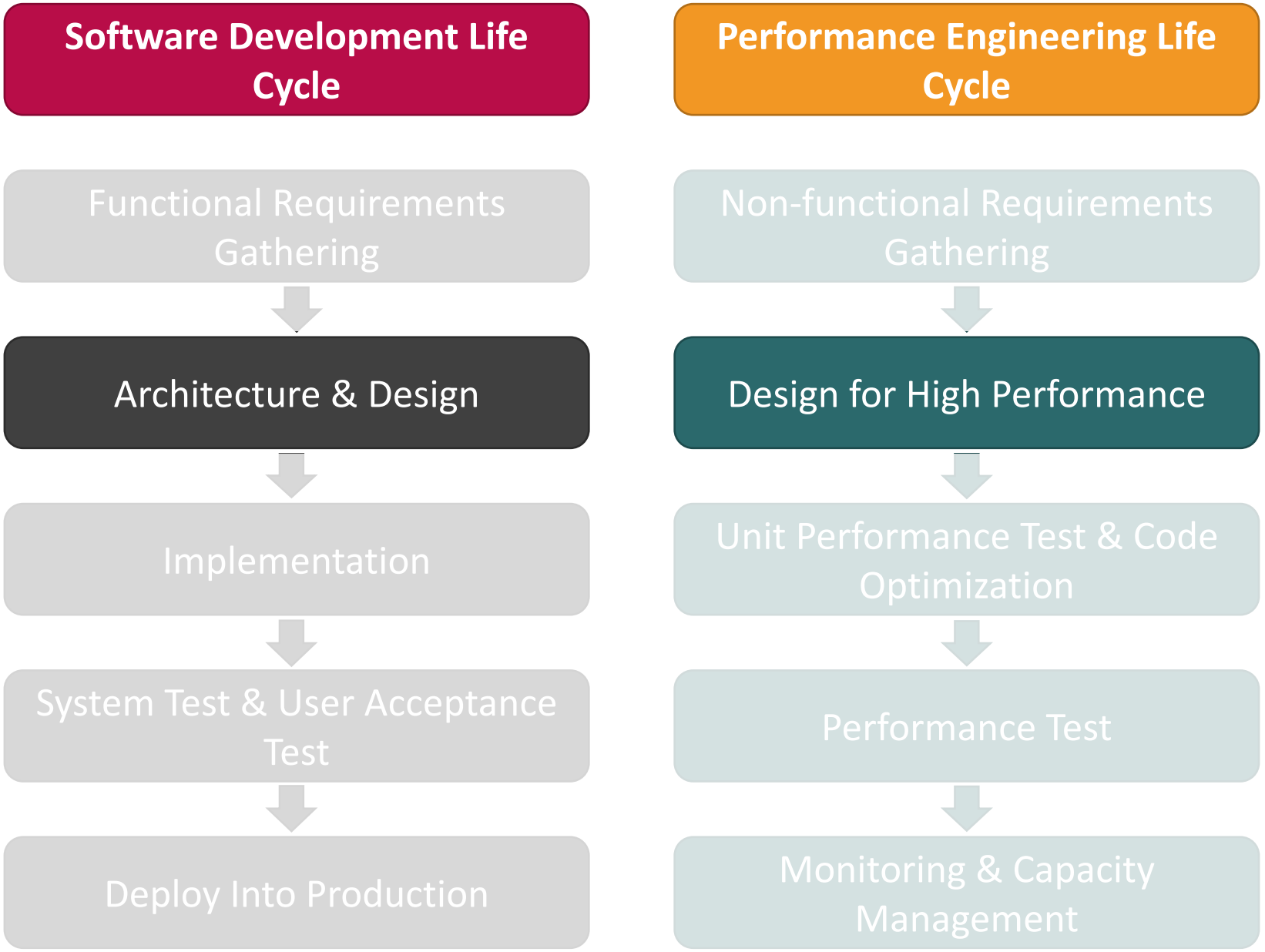
SPE: Objectives

- Increase revenue by ensuring the system processes all transactions in a timely manner.
- Eliminate delayed deployment due to performance issues.
- Eliminate unnecessary reengineering effort due to performance issues.
- Avoid additional and unnecessary costs of purchasing equipment.
- Reduce the increased costs of maintenance due to performance issues during production or *ad hoc* performance corrections.
- Reduce operational overhead to address system problems due to performance issues.
- Identify bottlenecks by simulating a prototype.
- Increase server capacity.


Software Development Life Cycle
Performance Engineering Life Cycle
Functional Requirements Gathering
Non-functional Requirements Gathering
Architecture & Design
Design for High Performance
Implementation
Unit Performance Test & Code Optimization
System Test & User Acceptance Test
Performance Test
Deploy Into Production
Monitoring & Capacity Management

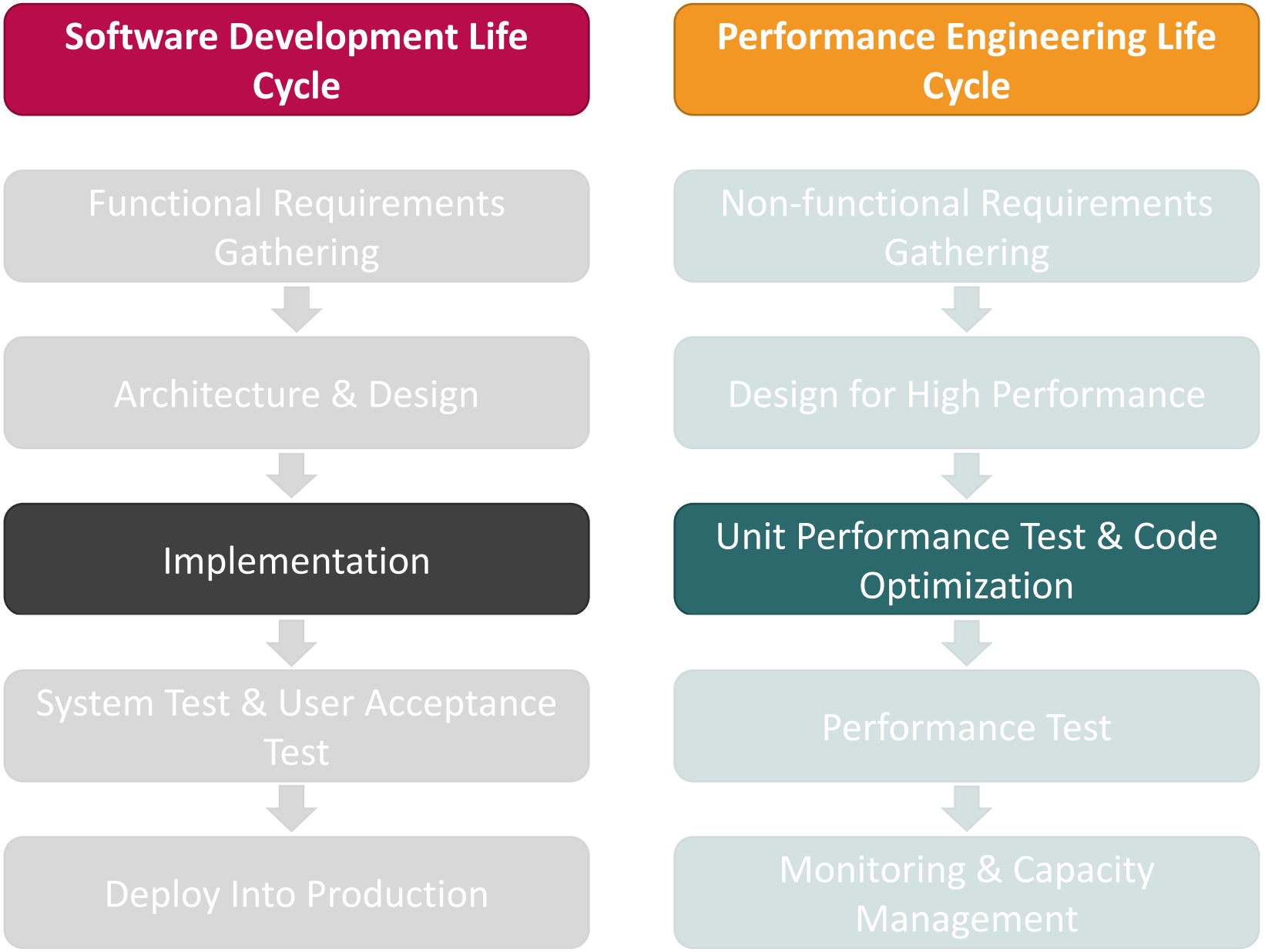
SPE: Requirements Phase

- Review business requirements and documentation.
  - Understand the business objectives and the platforms used to deliver them.
- Review production performance metrics of the current version if available.
- Determine non-functional requirements.
  - So that system performance goals can be set and measured against.
- Identify tools, resources and infrastructure.
  - Early identification allows budget and time allocation for installation and staff training.
- Confirm the consistency of the requirements with each other and the functional requirements.
  - Resolve conflicts between requirements.


Software Development Life Cycle
Functional Requirements Gathering
Architecture & Design
Implementation
System Test & User Acceptance Test
Deploy Into Production
Performance Engineering Life Cycle
Non-functional Requirements Gathering
Design for High Performance
Unit Performance Test & Code Optimization
Performance Test
Monitoring & Capacity Management

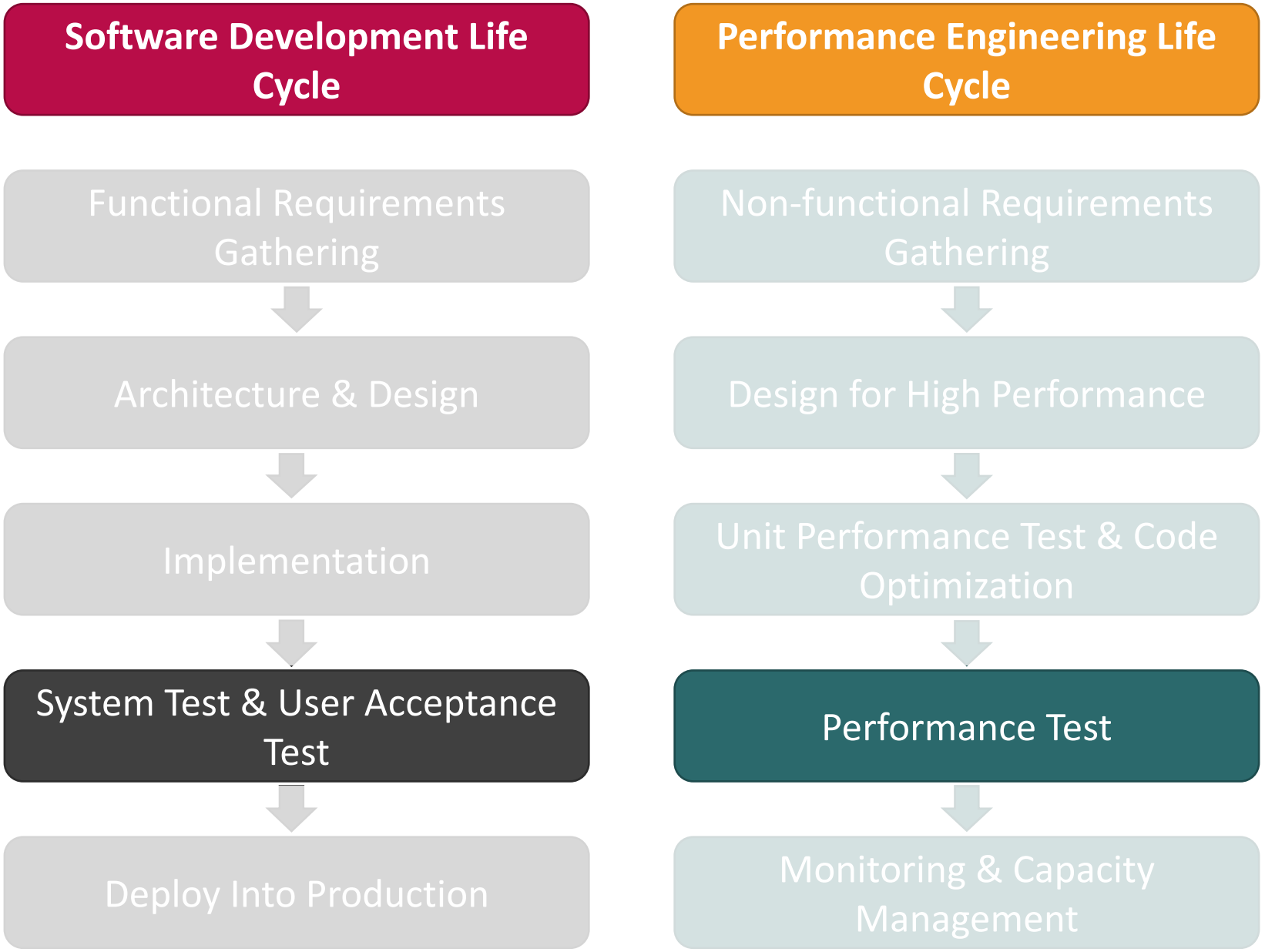
SPE: Architecture and Design

- Evaluate the alternatives.
  - Provide input from a performance perspective to the architecture being recommended.
- Determine the capacity of the required infrastructure.
  - By combining the non-functional requirements with the architecture design, determine the underlying infrastructure requirements.
- Define performance targets for developers.
  - Performance targets for the development teams across application components and tiers (used for unit performance tests).


Software Development Life Cycle
Functional Requirements Gathering
Architecture & Design
Implementation
System Test & User Acceptance Test
Deploy Into Production
Performance Engineering Life Cycle
Non-functional Requirements Gathering
Design for High Performance
Unit Performance Test & Code Optimization
Performance Test
Monitoring & Capacity Management

SPE: Implementation

- Monitor the development and unit performance testing.
- Develop workload models.
  - Business workload: how users will use the system to achieve business goals (e.g., Transactions per hour), including any peak load periods or regular cycles (e.g., quarterly).
  - Infrastructure workload: the workload on infrastructure resources (e.g. CPU, memory, network utilization etc.)
- Install and configure performance monitoring tools for the software and its infrastructure.


Software Development Life Cycle
Performance Engineering Life Cycle
Functional Requirements Gathering
Non-functional Requirements Gathering
Architecture & Design
Design for High Performance
Implementation
Unit Performance Test & Code Optimization
System Test & User Acceptance Test
Performance Test
Deploy Into Production
Monitoring & Capacity Management

SPE: Testing

- Create performance tests to simulate the workload model.
- Use the tests to validate the non-functional requirements.
- Identify application bottlenecks.
- Validate the impact of code and configuration changes on application performance.
  - Identify performance regressions

What is a performance regression?

Old version

version

Does the new version have worse performance than the old version?

Mozilla takes performance regression seriously!

Performance Regressions Policy

In this document:

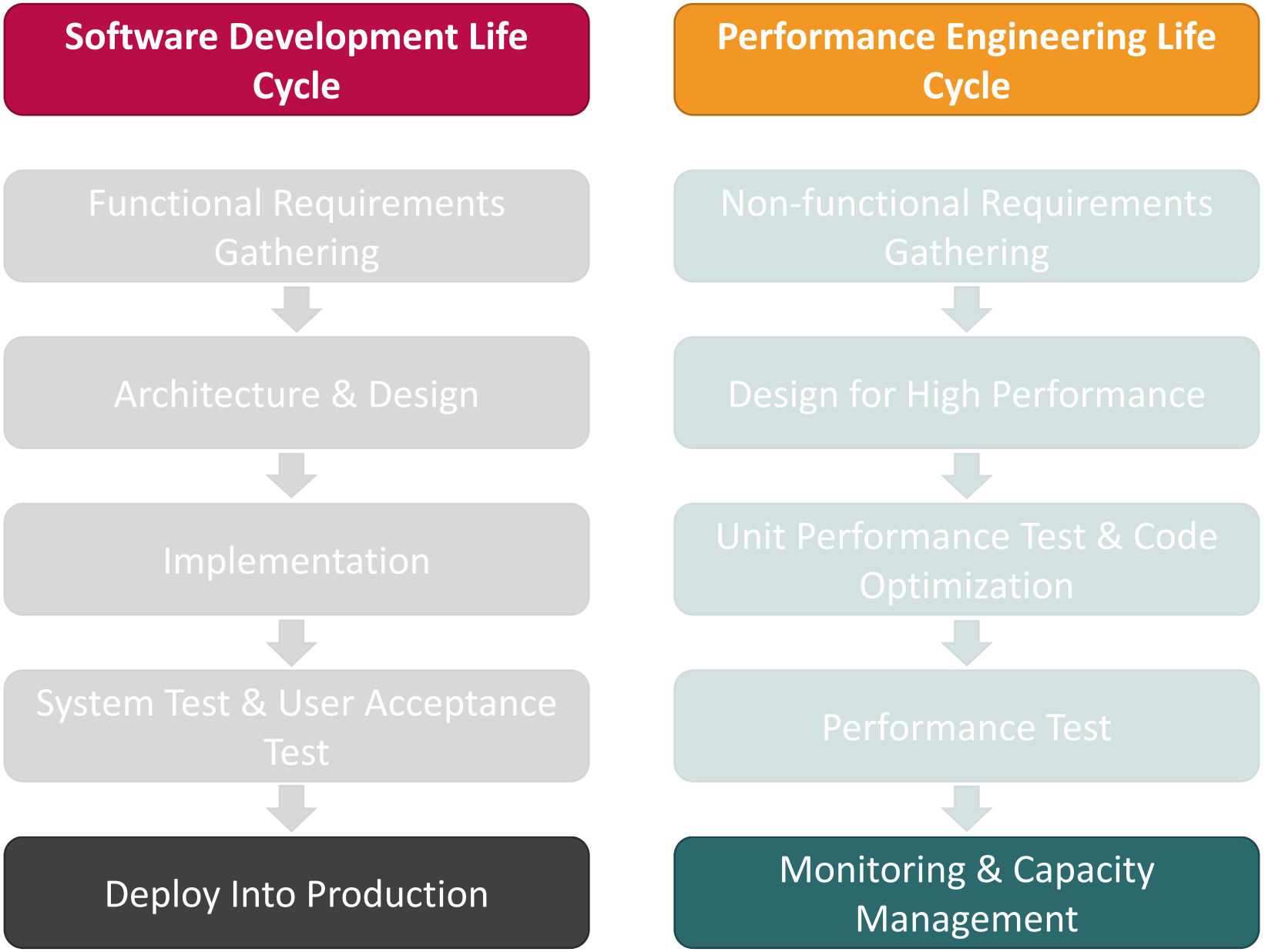
- Basic Policy: Patch authors will be notified with a bug of regressions and have 3 days to respond or get backed out.
- Requirements: Requirements for bugs.
- Acceptable Outcomes: Responses to regressions.
- Other Scenarios: Non standard regressions.

Basic Policy: Bug filed

Performance is a critical goal for Mozilla releases and the commercial products that will be

Mozilla takes performance regression seriously!

"We cannot allow performance regressions to go unnoticed or unresolved during our development cycles."


Software Development Life Cycle
Performance Engineering Life Cycle
Functional Requirements Gathering
Non-functional Requirements Gathering
Architecture & Design
Design for High Performance
Implementation
Unit Performance Test & Code Optimization
System Test & User Acceptance Test
Performance Test
Deploy Into Production
Monitoring & Capacity Management

SPE: Production

- Perform performance monitoring to continuously assess software performance, and to identify when the system is reaching its capacity.
- Perform capacity management to provide the required infrastructure capacity to sustain growth in business workloads.
- Provide production workload data to support the development of the next version.

Benefits of SPE

- Defining a clear set of non-functional requirements ensures successful development.
- The constant and early focus on system performance during all phases of development prevents late and costly changes in the future.
- Production performance monitoring maintains system performance and allows capacity to be expanded before it is exceeded.
- The proactive approach allows you to avoid problems and focus on development, not on constant problem solving.
- By successfully delivering a functional and performing system as required, the customer will receive full value.

Challenges of SPE

- By promoting time to market and budget constraints, the importance of SPE in the software lifecycle is reduced.
- The main challenge for an inefficient SPE is a knowledge gap between developers and quality experts.
  - This is also the reason for the difficulty of matching functional requirements with non-functional requirements (but not impossible).
- Performance is perceived by users.
  - Developers know the features but they cannot perceive the performance.
  - Quality experts know performance, but they don't know features.
- One recommended solution to bridge the gap is automation.
  - Eliminate the need for qualified people for manual construction of methods and models for performance.
  - Reduce time and effort for performance validation.

Performance Testing

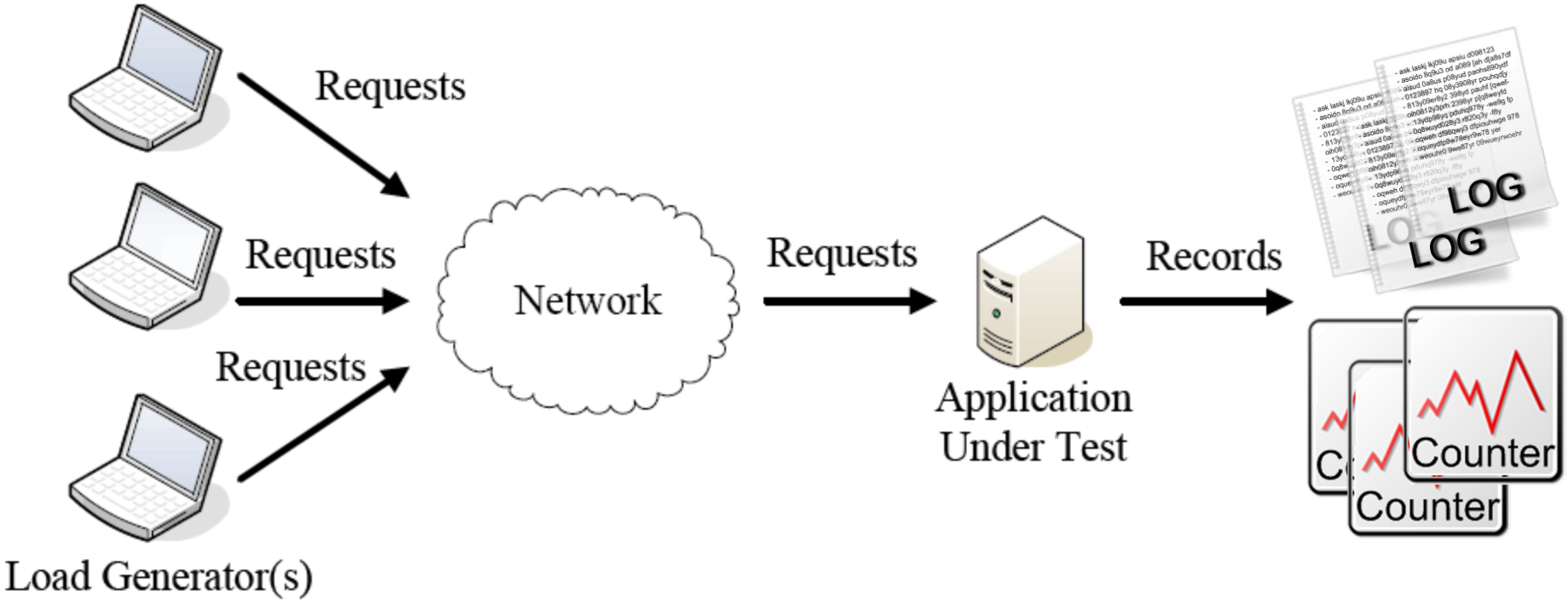
Performance Testing

- All the tests and methodologies to measure, verify and validate the performance of the system.
- It is part of SPE.
- Its objectives include:
  - Demonstrate system compliance with performance criteria.
  - Compare two systems to find the most efficient.
  - Measure and identify the components that cause the system to not perform well.

Testing: Types of tests

- **Load testing**: Tests the performance of the system under the expected load.
  - A number of users who perform a specified number of requests during a given period of time.
- **Stress testing**: Tests the limits of the system's capacity.
- **Endurance testing**: Tests the system under the expected load for a long time.
- **Spike testing**: Tests the reaction of the system by suddenly increasing or decreasing the load generated by a very large number of users
- **Capacity testing**: Tests the system to find the maximum capacity.
- **Configuration testing**: Test the effect of various configuration or configuration changes to the system.

Performance Testing



Test Design **Test Execution** **Test Analysis**

Mimics multiple users repeatedly performing the same tasks

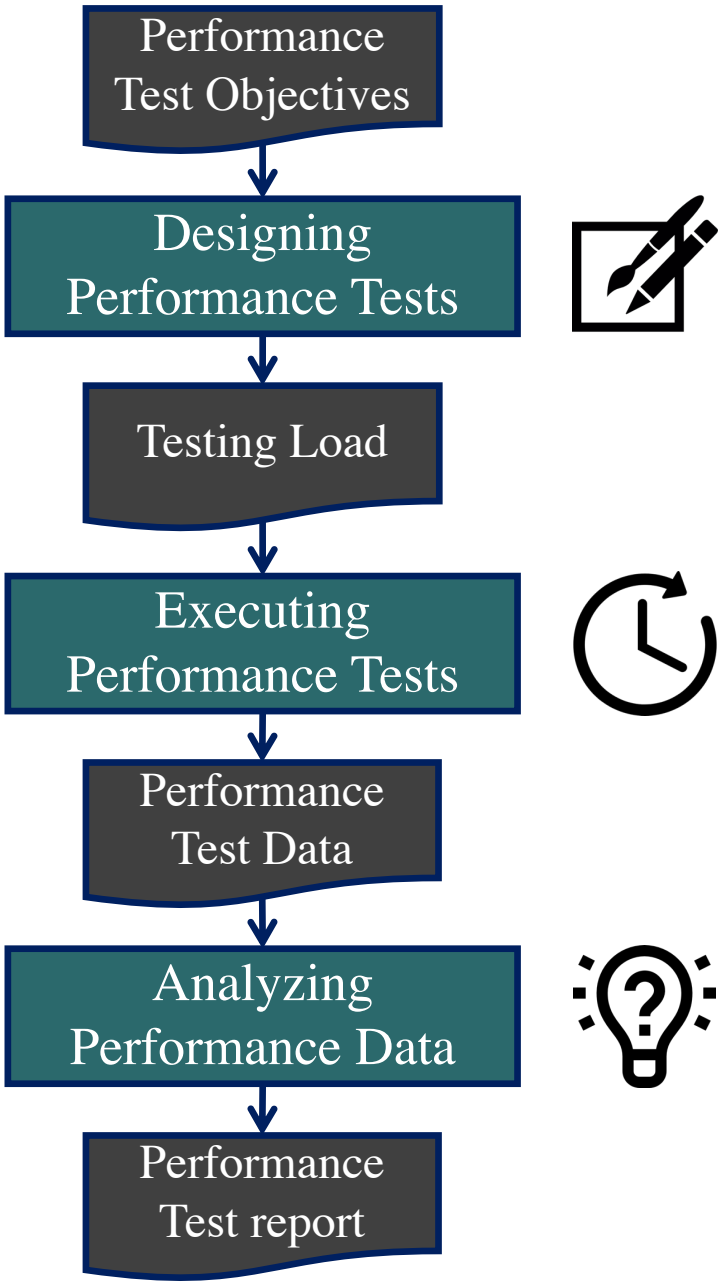
Take hours or even days

Produces GB/TB of data that must be analyzed

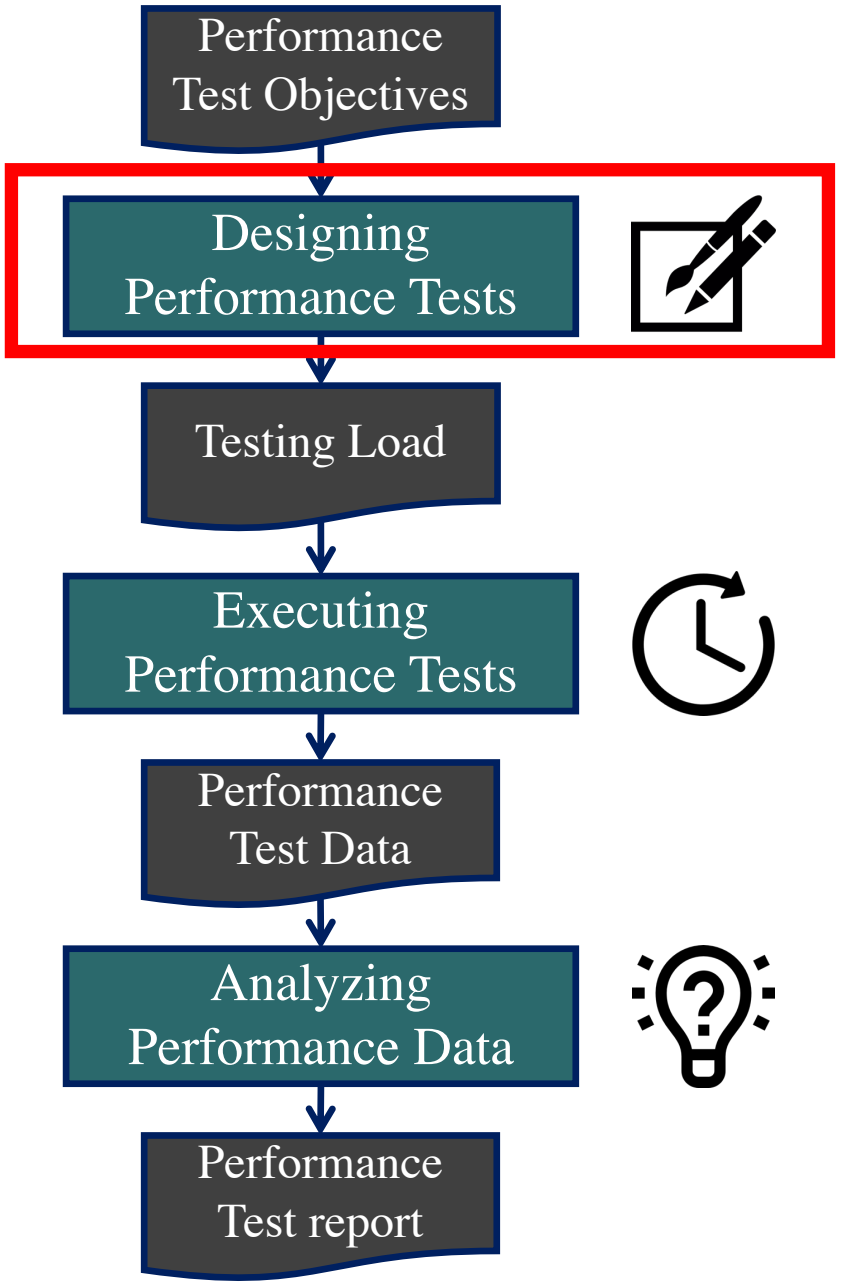
Testing: Errors

- Performance testing is the last step in development.
- More hardware fixes all performance issues.
- What works now, it will always work.
- One testing scenario is sufficient.
- Testing every part of the system equals testing the entire system.
- Developers are too experienced to need testing.
- Load testing is sufficient.

Performance Test Process



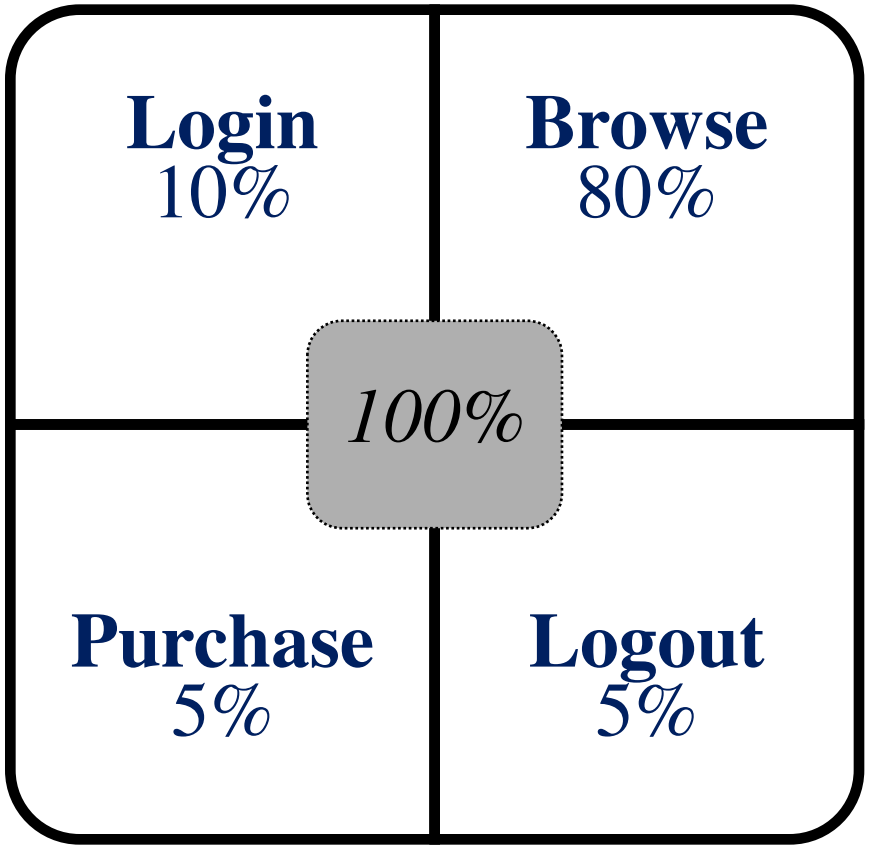
Performance Test Process



Designing Realistic Loads

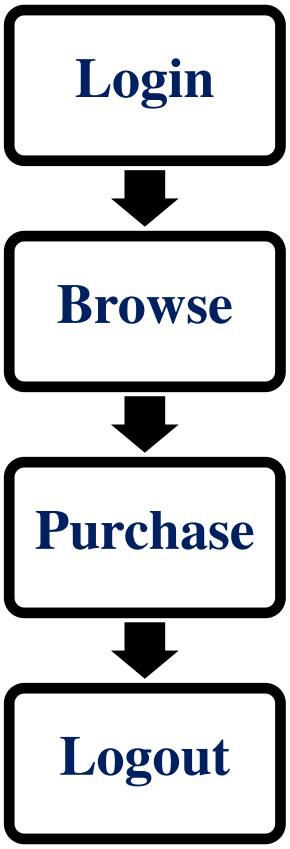
An E-Commerce System

Aggregate Workload



Steady Load, Step-wise load,
Extrapolated load

Use-Case



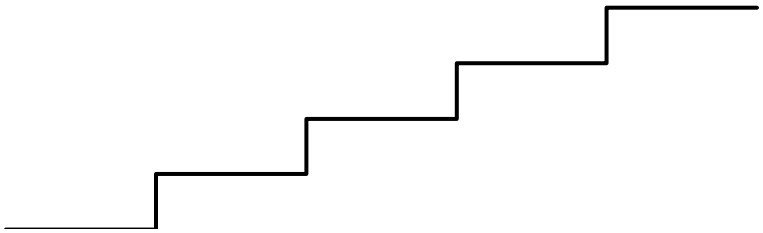
Load Derived from UML, Markov and
Stochastic Form-oriented Models

Aggregate Workload (1)

- Steady Load
  - Ease of measurement
  - Memory leaks?

[Bondi, CMG 2007]

- Step-wise Load
  - Same workload mix
  - Different workload intensity



[Hayes, CMG 2000]

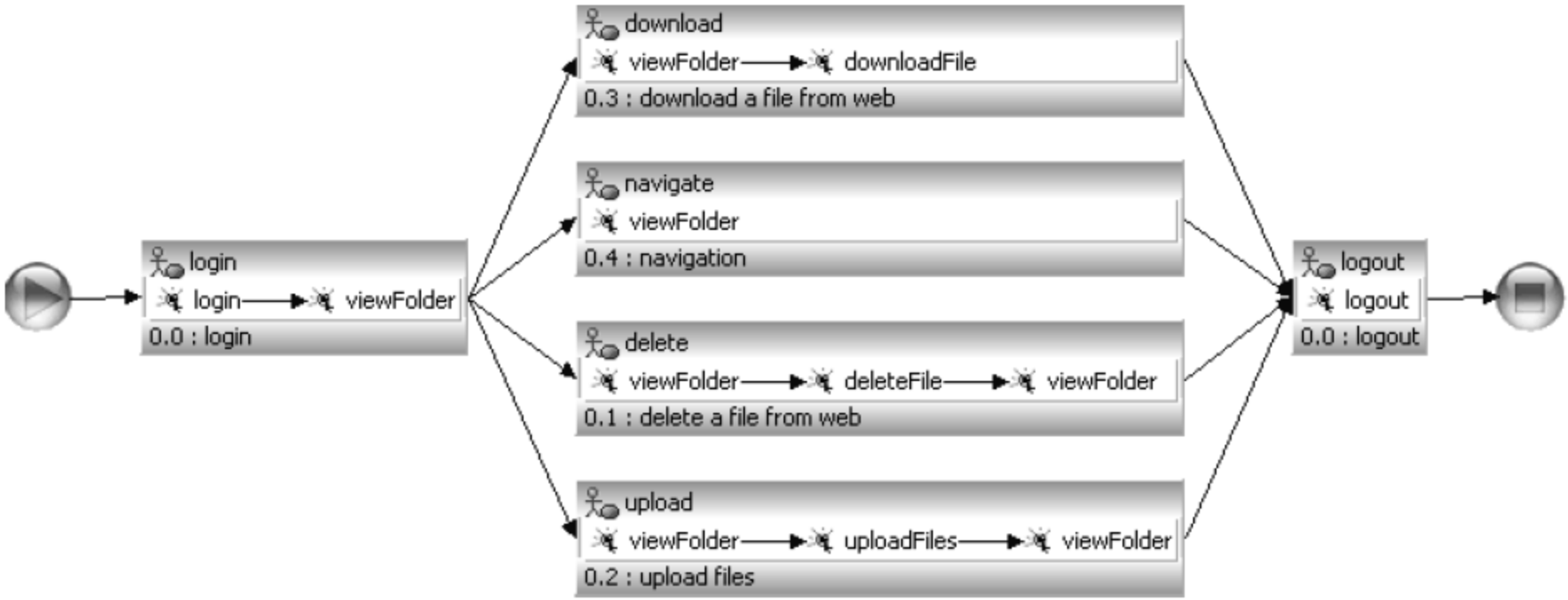
Derived the testing loads from historic data

Aggregate Workload (2)

- In case of missing past usage data, testing loads can be extrapolated from the following sources:
  - Beta-usage data
  - Interviews with domain experts
  - Competitors' data

[Barber, WSE 2004]

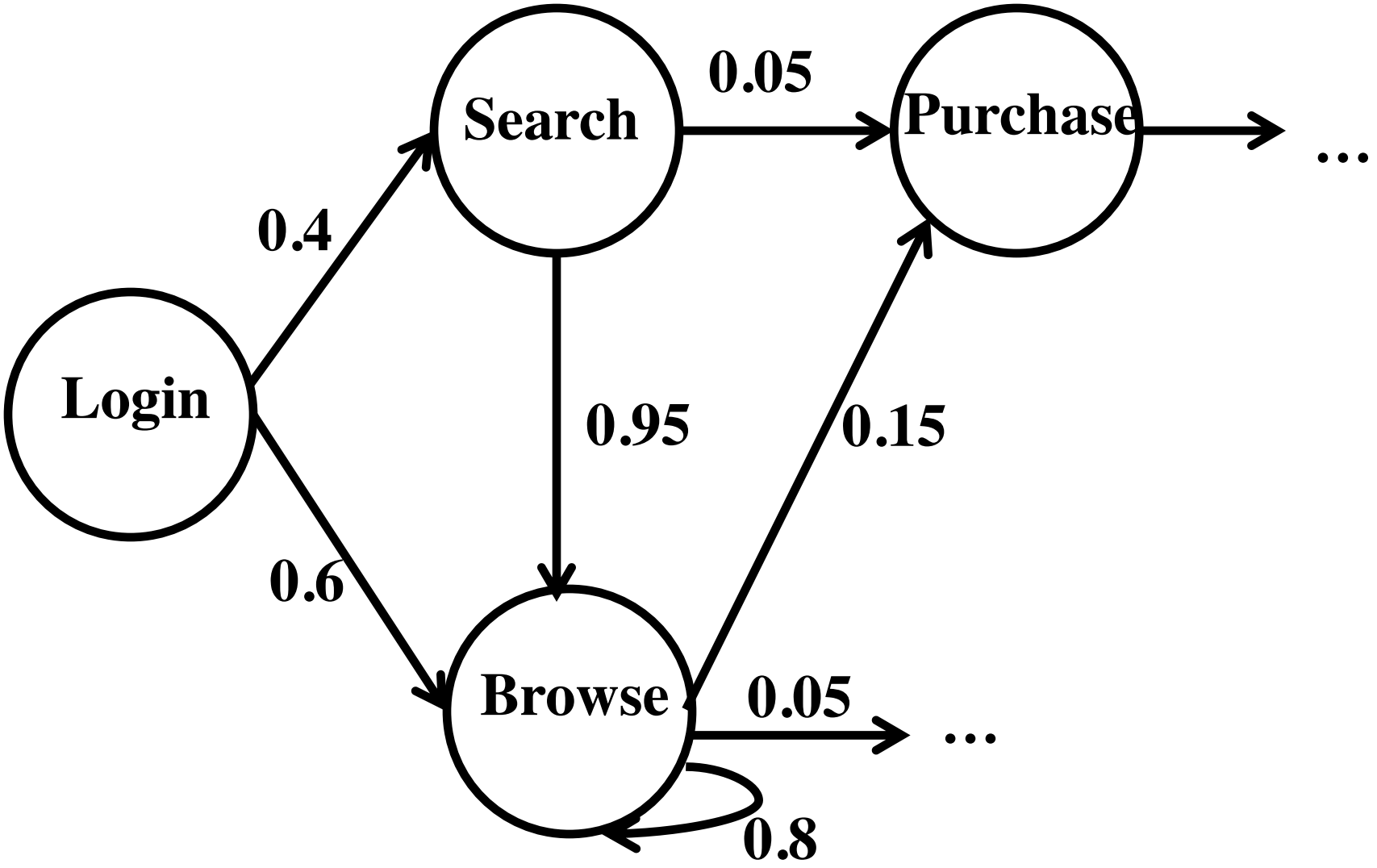
Use-Case (1)
- UML Diagrams



The RUG (Realistic Usage Model)
- derived based on UML use case diagrams

[Wang, ISPA 2004]

Use-Case (2)
- Markov Chain



Use-Case (2)
- Markov Chain

```
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dsbrowse.jsp?browsetype=actor&browse_category=&browse_actor=ANTHONY%20
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dsbrowse.jsp?browsetype=category&browse_category=11&browse_actor=&brow
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dslogin.jsp?username=user41&password=password HTTP/1.1" 200 2539 16
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dsbrowse.jsp?browsetype=actor&browse_category=&browse_actor=WILLIAM%20
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dsbrowse.jsp?browsetype=category&browse_category=15&browse_actor=&brow
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dsbrowse.jsp?browsetype=actor&browse_category=&browse_actor=HILARY%20G
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dsbrowse.jsp?browsetype=category&browse_category=6&browse_actor=&brows
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dsbrowse.jsp?browsetype=title&browse_category=&browse_actor=&browse_ti
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dspurchase.jsp?confirmpurchase=yes&customerid=5961&item=646&quan=3&ite
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dspurchase.jsp?confirmpurchase=yes&customerid=41&item=4544&quan=1&item
192.168.0.1 - [22/Apr/2014:00:32:29 -0400] "GET /dslogin.jsp?username=user3614&password=password HTTP/1.1" 200 728 6
192.168.0.1 - [22/Apr/2014:00:32:29 -0400] "GET /dsbrowse.jsp?browsetype=title&browse_category=&browse_actor=&browse_ti
192.168.0.1 - [22/Apr/2014:00:32:29 -0400] "GET /dsbrowse.jsp?browsetype=actor&browse_category=&browse_actor=ELLEN%20GA
192.168.0.1 - [22/Apr/2014:00:32:29 -0400] "GET /dsbrowse.jsp?browsetype=category&browse_category=9&browse_actor=&brows
192.168.0.1 - [22/Apr/2014:00:32:29 -0400] "GET /dsbrowse.jsp?browsetype=actor&browse_category=&browse_actor=ANGELINA%2
192.168.0.1 - [22/Apr/2014:00:32:29 -0400] "GET /dsbrowse.jsp?browsetype=actor&browse_category=&browse_actor=JULIA%20TA
192.168.0.1 - [22/Apr/2014:00:32:29 -0400] "GET /dspurchase.jsp?confirmpurchase=yes&customerid=3614&item=4717&quan=2&it
192.168.0.1 - [22/Apr/2014:00:32:31 -0400] "GET /dslogin.jsp?username=user13337&password=password HTTP/1.1" 200 1960 9
192.168.0.1 - [22/Apr/2014:00:32:31 -0400] "GET /dsbrowse.jsp?browsetype=title&browse_category=&browse_actor=&browse_ti
192.168.0.1 - [22/Apr/2014:00:32:31 -0400] "GET /dspurchase.jsp?confirmpurchase=yes&customerid=13337&item=322&quan=2&it
192.168.0.1 - [22/Apr/2014:00:32:35 -0400] "GET /dslogin.jsp?username=user5414&password=password HTTP/1.1" 200 2579 10
192.168.0.1 - [22/Apr/2014:00:32:35 -0400] "GET /dsbrowse.jsp?browsetype=actor&browse_category=&browse_actor=GRACE%20BR
192.168.0.1 - [22/Apr/2014:00:32:35 -0400] "GET /dspurchase.jsp?confirmpurchase=yes&customerid=5414&item=198&quan=3&ite
192.168.0.1 - [22/Apr/2014:00:32:35 -0400] "GET /dsnewcustomer.jsp?firstname=RHVSQS&lastname=EBFMQDBVNM&address1=909823
192.168.0.1 - [22/Apr/2014:00:32:35 -0400] "GET /dsbrowse.jsp?browsetype=title&browse_category=&browse_actor=&browse_ti
192.168.0.1 - [22/Apr/2014:00:32:35 -0400] "GET /dspurchase.jsp?confirmpurchase=yes&customerid=20001&item=7868&quan=3&i
192.168.0.1 - [22/Apr/2014:00:32:36 -0400] "GET /dslogin.jsp?username=user13713&password=password HTTP/1.1" 200 729 6
192.168.0.1 - [22/Apr/2014:00:32:36 -0400] "GET /dsbrowse.jsp?browsetype=category&browse_category=9&browse_actor=&brows
192.168.0.1 - [22/Apr/2014:00:32:36 -0400] "GET /dspurchase.jsp?confirmpurchase=yes&customerid=13713&item=493&quan=3&it
192.168.0.1 - [22/Apr/2014:00:32:41 -0400] "GET /dsloqin.jsp?username=user9011&password=password HTTP/1.1" 200 728 6
```

web access logs for the past few months

Use-Case (2)
- Markov Chain

192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dsbrowse.jsp?browsetype=title&browse_category=&browse_actor=&browse_title=HOLY%20AUTUMN&limit_num=8&customerid=41 HTTP/1.1" 200 4073 10

192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dspurchase.jsp?confirmpurchase=yes&customerid=5961&item=646&quan=3&item=2551&quan=1&item=45&quan=3&item=9700&quan=2&item=1566&quan=3&item=4509&quan=3&item=5940&quan=2 HTTP/1.1" 200 3049 177

192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /dspurchase.jsp?confirmpurchase=yes&customerid=41&item=4544&quan=1&item=6970&quan=3&item=5237&quan=2&item=650&quan=1&item=2449&quan=1 HTTP/1.1" 200 2515 113

Web Access Logs

Use-Case (2)
- Markov Chain

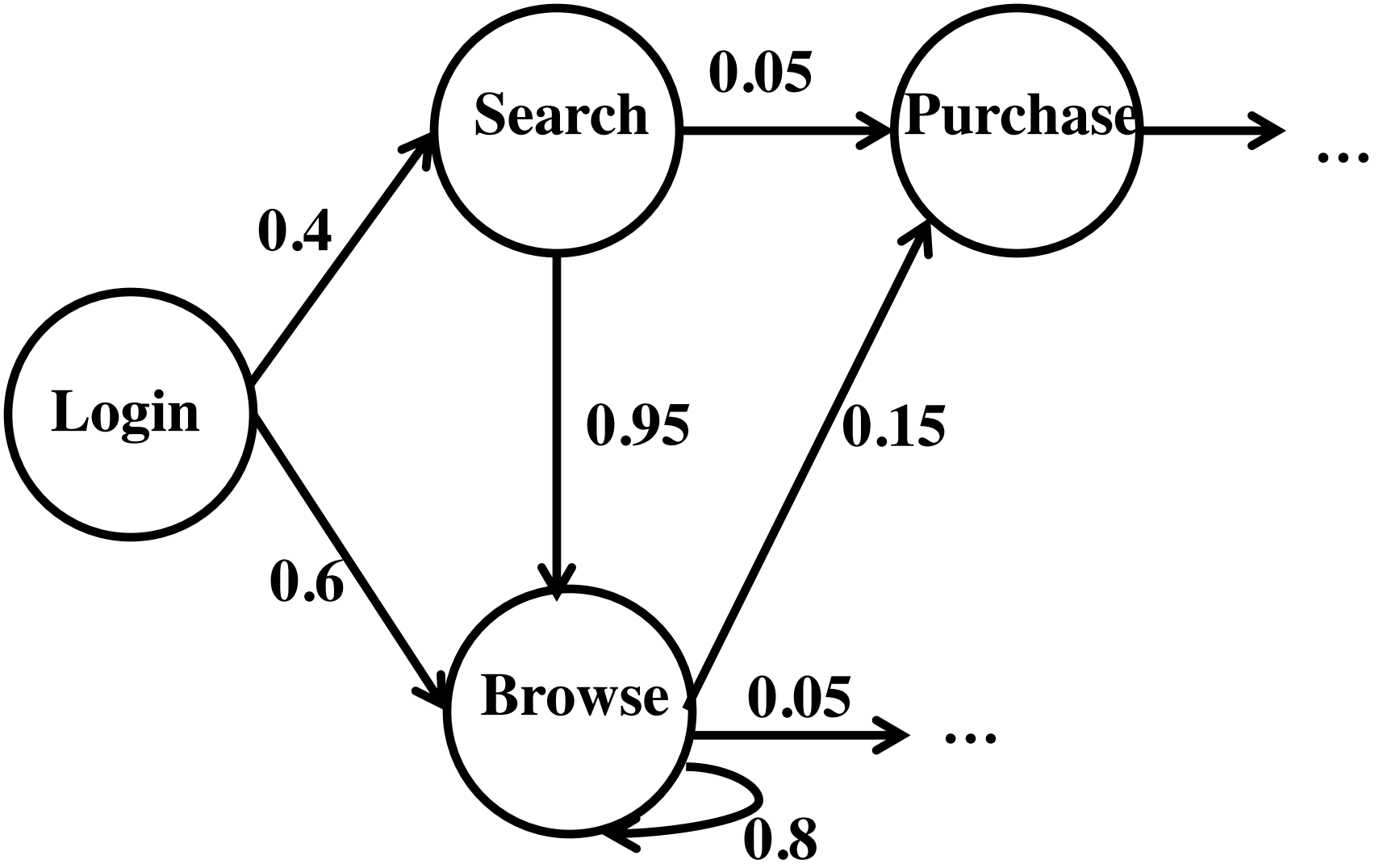
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /**dsbrowse.jsp**?browsetype=title&browse_category=&browse_actor=&browse_title=HOLY%20AUTUMN&limit_num=8&**customerid=41** HTTP/1.1" 200 4073 10

192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /**dspurchase.jsp**?confirmpurchase=yes&**customerid=5961**&item=646&quan=3&item=2551&quan=1&item=45&quan=3&item=9700&quan=2&item=1566&quan=3&item=4509&quan=3&item=5940&quan=2 HTTP/1.1" 200 3049 177

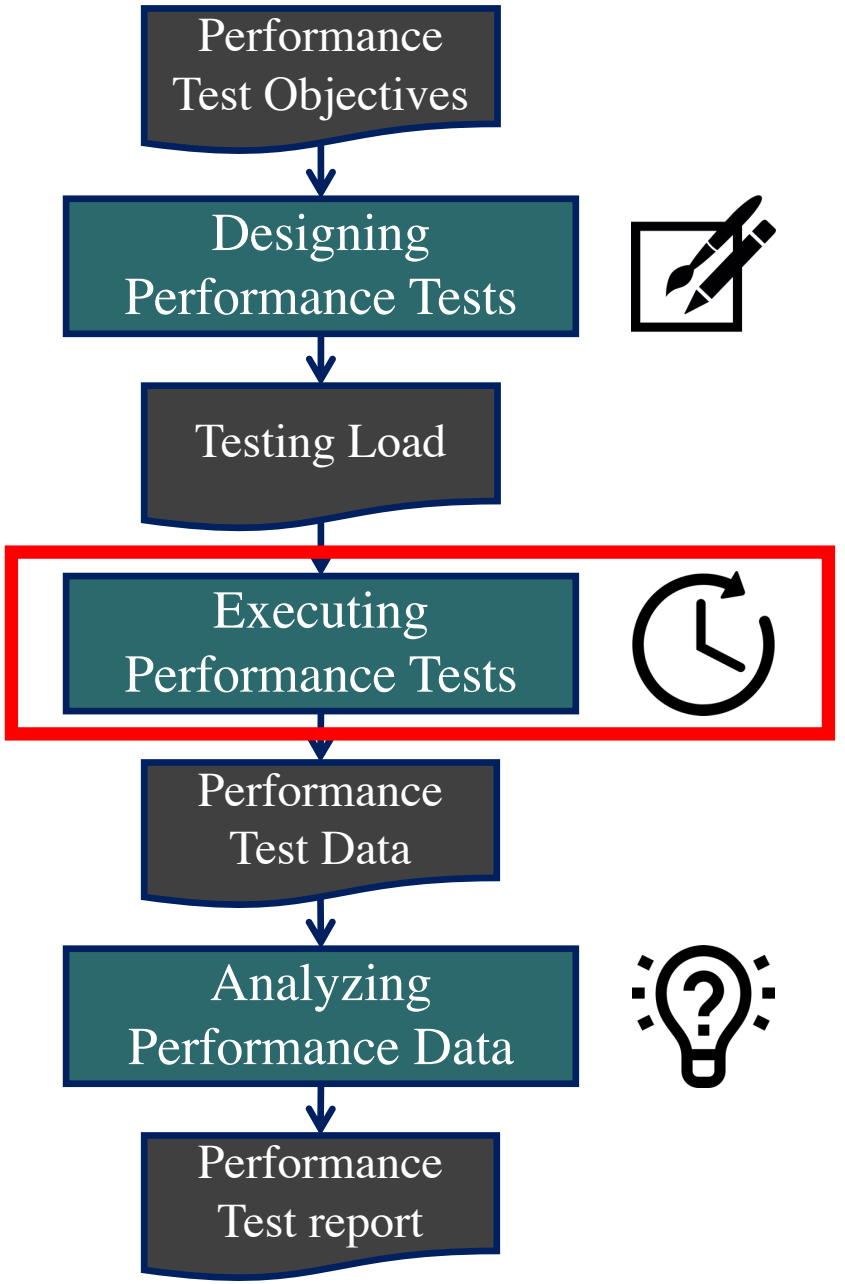
192.168.0.1 - [22/Apr/2014:00:32:25 -0400] "GET /**dspurchase.jsp**?confirmpurchase=yes&**customerid=41**&item=4544&quan=1&item=6970&quan=3&item=5237&quan=2&item=650&quan=1&item=2449&quan=1 HTTP/1.1" 200 2515 113

For customer 41: browse -> purchase

Use-Case (2)
- Markov Chain



Performance Test Process



Live-user Based Test Execution

- Coordinated live-user testing
- Users are selected based on different testing criteria (e.g., locations, browser types, etc.)

- Reflects realistic user behavior
- Obtain real user feedbacks on acceptable performance and functional correctness

- Hard to scale (e.g., limited testing time)
- Limited test complexity due to manual coordination

uTest®

Driver-based Test Execution

- 👍 Easy to automate
- 👍 Scale to large number of requests

- 👎 Load driver configurations
- 👎 Hard to track some system behavior (e.g., audio quality or image display)

- Specialized Benchmarking tools (e.g., LoadGen)
- Centralized Load Drivers (e.g, LoadRunner, WebLoad)
  - Easy to control load, but hard to scale (limited to a machine's memory)
- Peer-to-peer Load Drivers (e.g., JMeter, PeerUnit)
  - Easy to scale, but hard to control load

APACHE JMeter™

Three General Aspects When Executing a Load Test

Test Setup

- System Deployment
- Test Execution Setup

Load Generation and Termination

- Static Configuration
- Dynamic Feedback
- Deterministic

Test Monitoring and Data Collection

- Metrics and Logs

Test Execution Setup

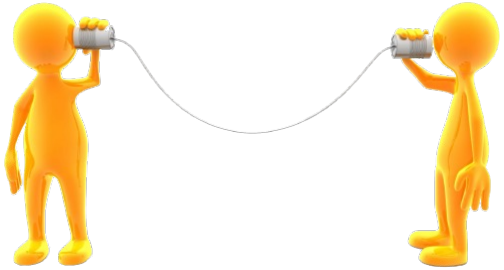
- Live-user-based executions
  - Tester recruitment, setup and training
- Driver-based executions
  - Programming
  - Store-and-replay configuration
  - Model configurations (e.g., Markov chain for JMeter as an extension)

Load Generation and Termination

Static Configuration

- Timer-Driven
- Counter-Driven
- Statistic-Driven

Dynamic Feedback

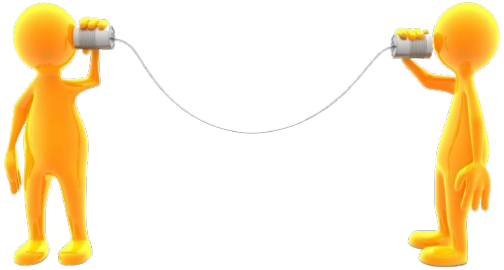


- Dynamically steer the testing loads based on system feedback

Load Generation and Termination

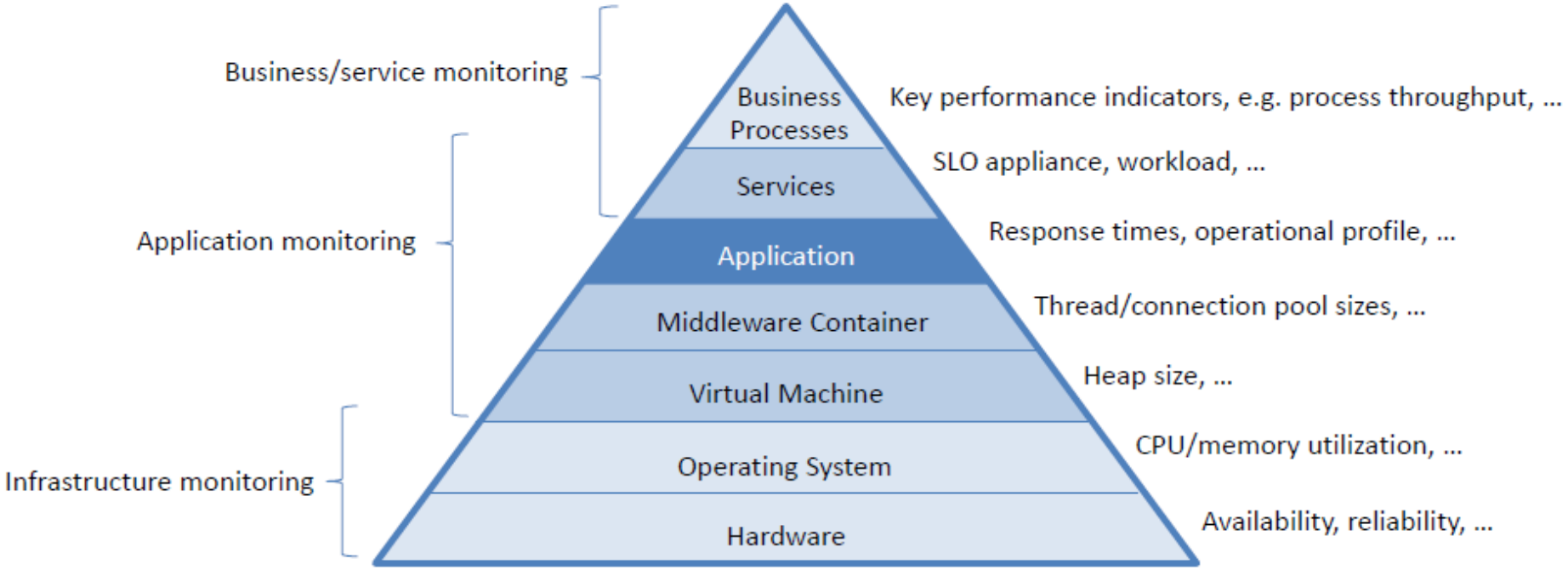
Static Configuration

Dynamic Feedback

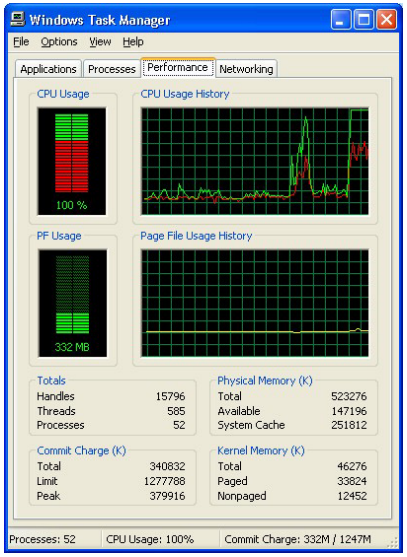


	Live-user Based	Driver Based
Static	✓	✓
Dynamic	×	✓

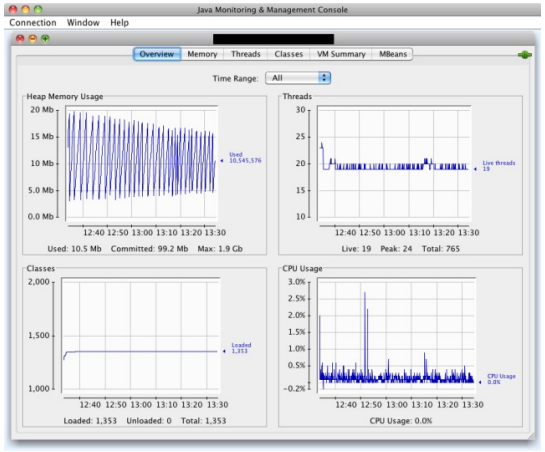
Performance Monitoring



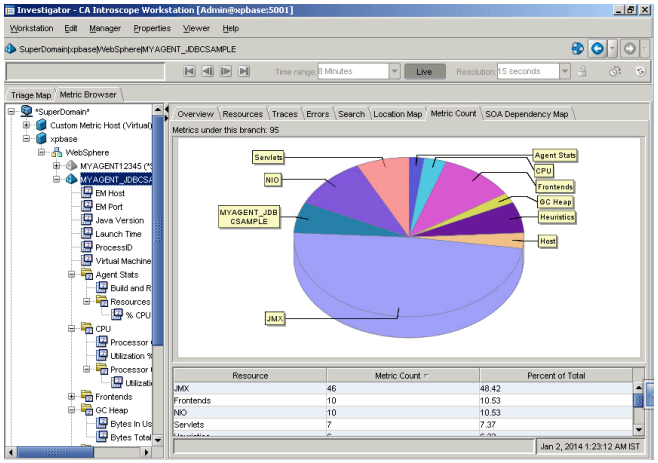
Test Monitoring Tools



Task Manager

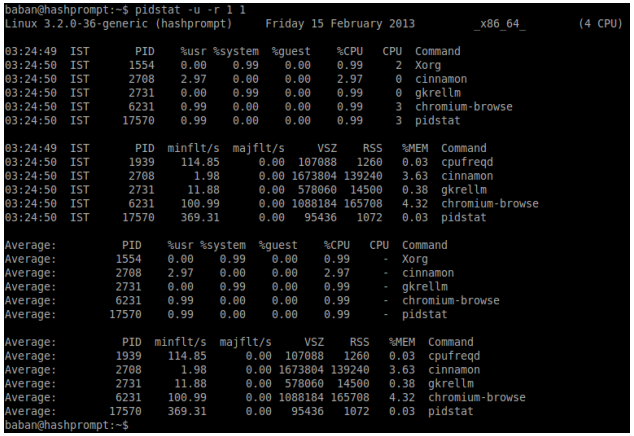


JConsole



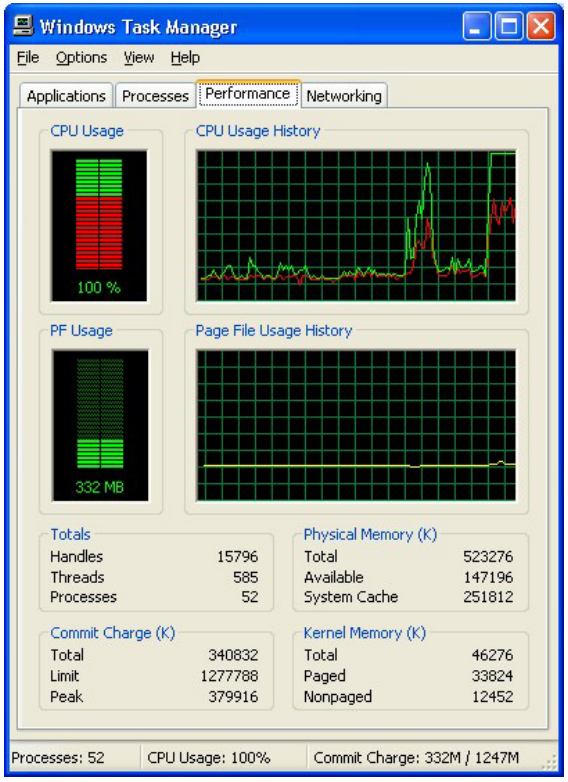
CA Willy

App Dynamics

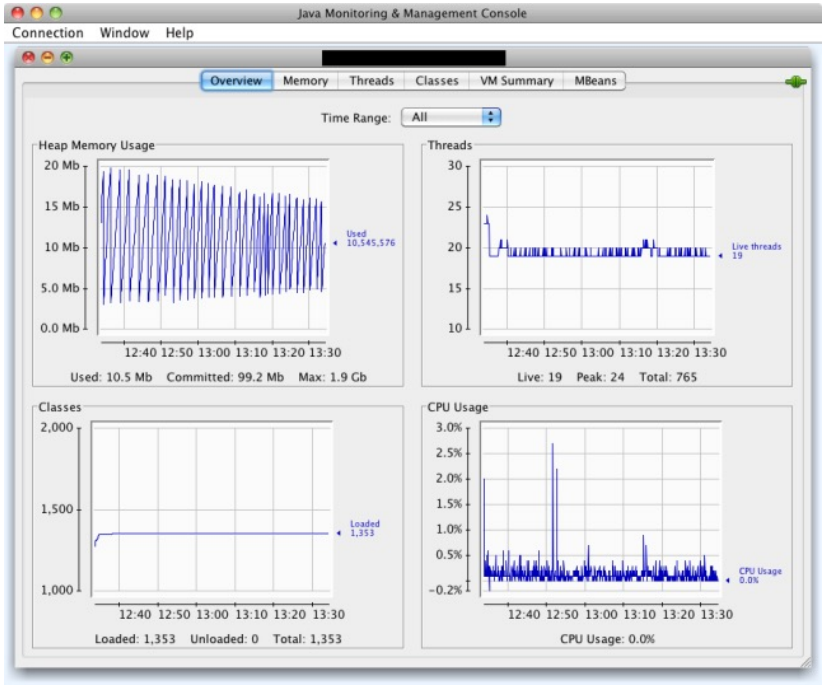


pidstat

Agent-less Monitoring Examples



Task Manager

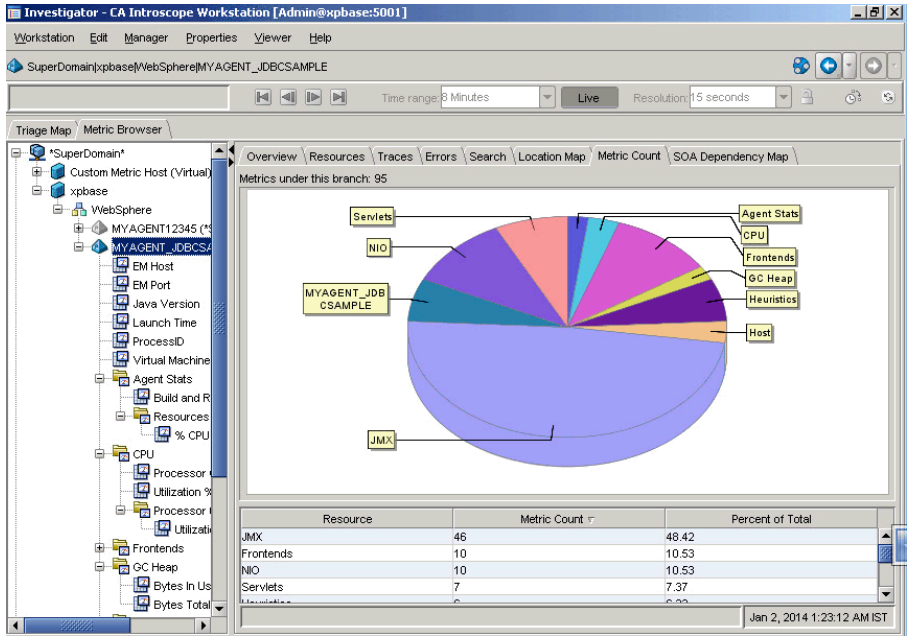


JConsole

PerfMon (Windows), sysstat (Linux), top

Agent-based Monitoring Examples

App Dynamics



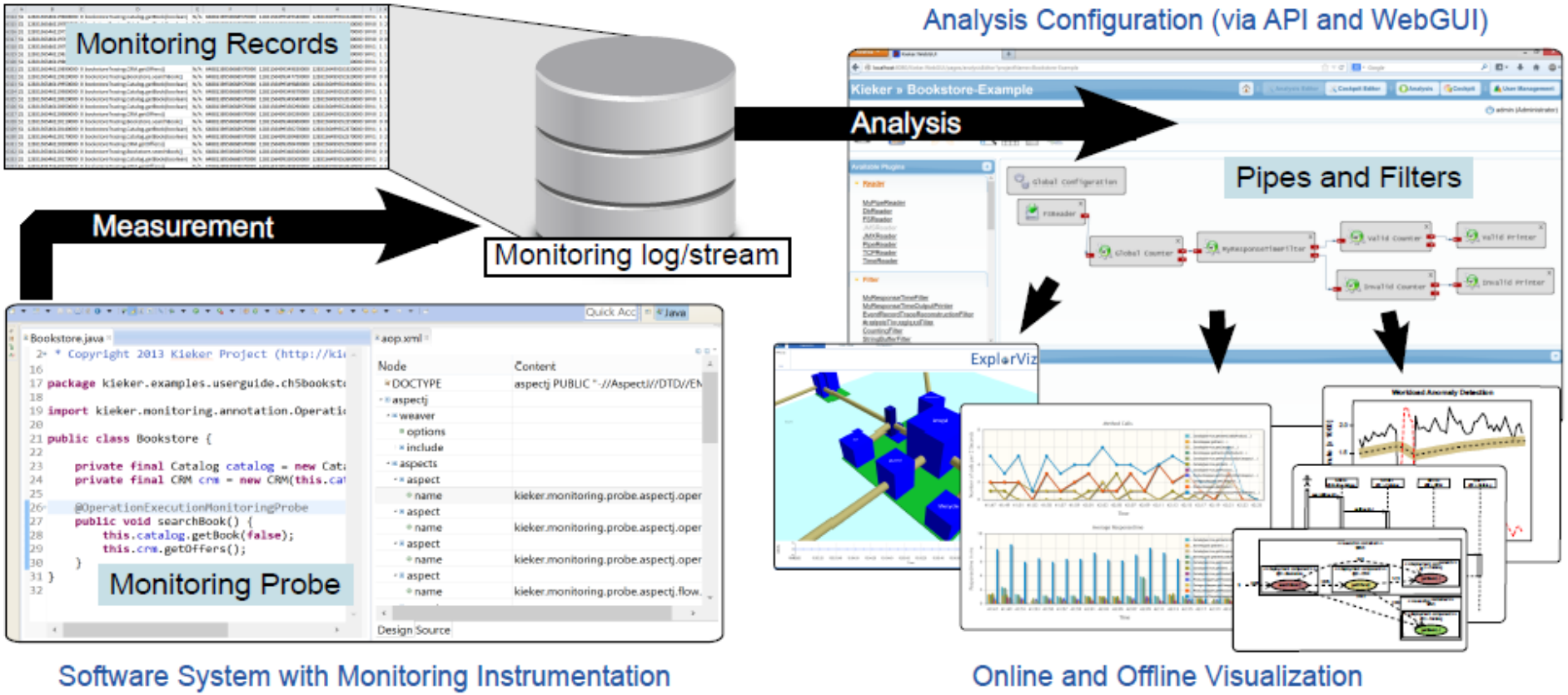
CA Willy

Dell FogLight, New Relic

Application Performance Monitoring (APM)

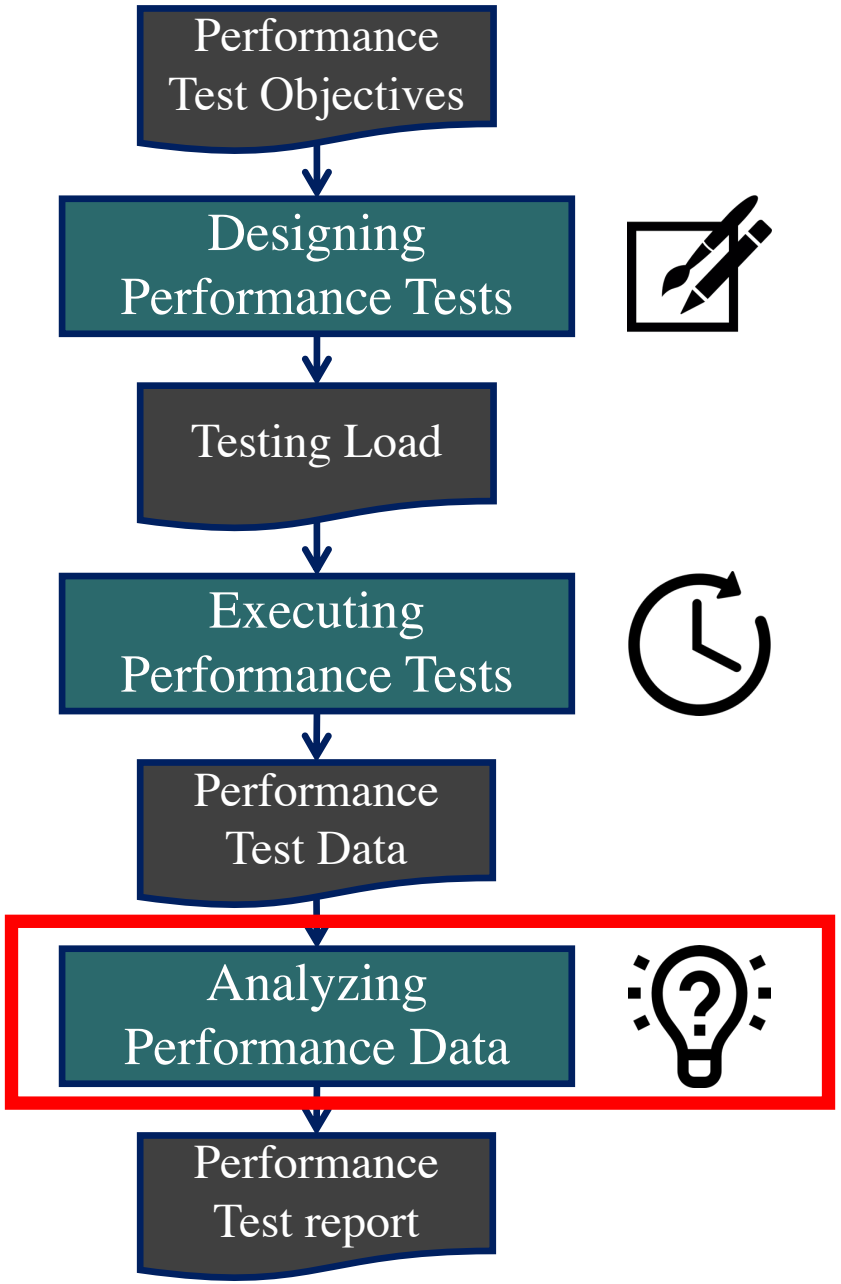
- Commerical Products:
  - AppDynamics, Compuware Dynatrace, …
- Open-Source:
  - Kieker http://kieker-monitoring.net/

Kieker Monitoring Framework



https://kieker-monitoring.net/live-demo/

Performance Test Process



Sample Counters

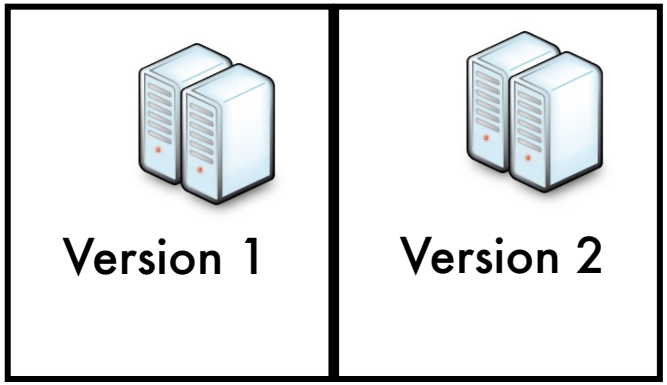
	A	B	C	D	E
1	Time	Disk Reads/sec	Disk Writes/sec	Page Faults/sec	Memory
2	2/29/08 16:58	0.049986394	0.000723659	0.003876542	3534848
3	2/29/08 17:01	0	0	0	3534848
4	2/29/08 17:04	0.060612225	0.027551011	0.016530607	3534848
5	2/29/08 17:07	0	0	0	3534848
6	2/29/08 17:10	0	0	0	3534848
7	2/29/08 17:13	0.060733302	0.027606046	0.016563628	3534848
8	2/29/08 17:16	0	0	0	3534848
9	2/29/08 17:19	0.060727442	0.027603383	0.01656203	3534848
10	2/29/08 17:22	0	0	0	3534848
11	2/29/08 17:25	0	0	0	3534848
12	2/29/08 17:28	0	0	0	3534848
13	2/29/08 17:31	0	0	0	3534848
14	2/29/08 17:34	0.121368621	0.055167555	0.038617289	3534848
15	2/29/08 17:37	0	0	0	3534848
16	2/29/08 17:40	0	0	0	3534848
17	2/29/08 17:43	0	0	0	3534848
18	2/29/08 17:46	0	0	0	3534848
19	2/29/08 17:49	0	0	0	3534848
20	2/29/08 17:52	0	0	0	3534848
21	2/29/08 17:55	0.121392912	0.055178596	0.033107158	3534848
22	2/29/08 17:58	0.060592703	0.027542138	0.02203371	3534848

Sample Execution Logs

#	Log Lines
1	time=1, thread=1, session=1, receiving new user registration request
2	time=1, thread=1, session=1, inserting user information to the database
3	time=1, thread=2, session=2, user=Jack, browse catalog=novels
4	time=1, thread=2, session=2, user=Jack, sending search queries to the database
5	time=3, thread=1, session=1, user=Tom, registration completed, sending confirmation email to the user
6	time=3, thread=2, session=2, database connection error: session timeout
7	time=4, thread=1, session=1, fail to send the confirmation email, number of retry = 1
8	time=6, thread=2, session=2, user=Jack, successfully retrieved data from the database
9	time=7, thread=2, system health check
10	time=8, thread=1, session=1, registration email sent successfully to user=Tom
11	time=9, thread=2, session=3, user=Tom, browse catalog=travel
12	time=10, thread=2, session=3, user=Tom, sending search queries to the database
13	time=10, thread=3, session=4, user=Jim, updating user profile
14	time=11, thread=3, session=4, user=Jim, database error: deadlock

Comparing with thresholds or reference versions

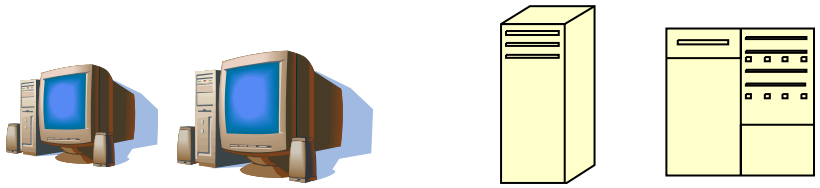
Comparing with threshold from requirement



Comparing with prior version

Comparing Alternatives

- Comparing one alternative with a threshold
- Comparing two alternatives
  - Non-corresponding measurements
  - Before-and-after comparisons
- Comparing proportions



Comparing one sample with a threshold

- Motivation
  - Is there a statistically significant difference between the performance of a system and a threshold?
- Assume there is one set of measurements (sample) corresponding to the alternative
- Example: **One-sample *t*-test**

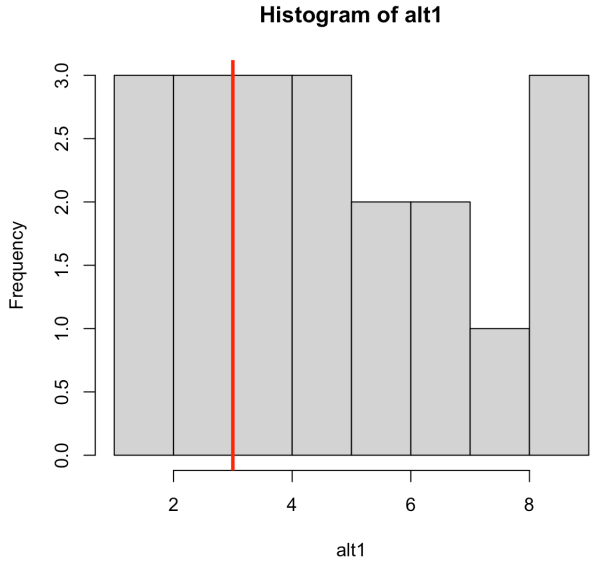
One Sample t-test with R

```
> alt1 <- c(3,7,1,9,3,4,1,2,6,7,5,8,5,9,4,6,4,3,9,5)
> thre <- 3
> t.test(alt1, mu=thre)

        One Sample t-test

data:  alt1
t = 3.604, df = 19, p-value = 0.001891
alternative hypothesis: true mean is not equal to 3
95 percent confidence interval:
 3.859453 6.240547
sample estimates:
mean of x
     5.05

> hist(alt1)
> abline(v=thre, col="red",lwd=3)
```



Comparing Two Alternatives

- Motivation
  - Is there a statistically significant difference between two systems?
  - Does a change made to a system have a statistically significant impact on its performance?
- Assume there are two sets of measurements (samples) corresponding to the two alternatives
- Will distinguish between two cases:
  - **Non-corresponding measurements (unpaired observations)**
    - The two sets of measurements (samples) are independent
  - **Before-and-after comparisons (paired observations)**
    - The two sets of measurements (samples) are not independent

Non-Corresponding Measurements

Assumptions

- Measurements form two independent samples
- Alternative 1: Alternative 2:
- Measurements within each set are independent and identically distributed (IID) random variables with variances, resp.

Example: Independent (unpaired) two-sample *t*-test

Two Sample t-test with R

```
> alt1<-c(3,7,1,9,3,4,1,2,6,7,5,8,5,9,4,6,4,3,9,5)
> alt2<-c(3,1,2,4,5,2,2,5,3,2,3,4,2,3,5,4,3,1,3,2)
> t.test(alt1,alt2)

  Welch Two Sample t-test

data:  alt1 and alt2
t = 3.3215, df = 27.478, p-value = 0.002539
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.8037895 3.3962105
sample estimates:
mean of x mean of y
     5.05      2.95

> par(mfrow=c(1,2))
> hist(alt1)
> hist(alt2)
```

Before-and-After Comparisons

- Assumptions
  - The two sets of measurements (samples) are not independent
  - Measurements can be grouped into corresponding pairs (b_i, a_i)
  - b_i = "before" measurement, a_i = "after" measurement
  - The set of differences $d_i = b_i - a_i$ are independent and identically distributed (IID) random variables (sample)
- Examples scenarios
  - The effect of an optimization applied to a set of systems

    Two corresponding measurements per system
  - A set of randomly selected benchmarks run on two systems

    Two corresponding measurements per benchmark

Example: **Dependent (paired) two-sample t-test**

Paired t-test with R

```
> before <- c(20,18,19,22,17,20,19,16,21,17,23,18)
> after <- c(22,19,17,18,21,23,19,20,22,20,27,24)
> t.test(before, after, paired=TRUE)

  Paired t-test
data:  before and after
t = -2.2496, df = 11, p-value = 0.04592
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -3.6270234 -0.0396433
sample estimates:
mean of the differences
              -1.833333

> t.test(before, after, paired=TRUE, conf.level = 0.96)
```

Comparing Proportions

Counting the number of times several events occur in two systems.

Want to compare the fraction of time a particular event occurs:

$X_1 = \#\text{occurances of the event in system 1}, \quad n_1 = \text{total } \#\text{events in system 1}$

$X_2 = \#\text{occurances of the event in system 2}, \quad n_2 = \text{total } \#\text{events in system 2}$

Example: **Two-proportion Z-test**

Comparing Proportions with R

```
> total <- c(1300203, 999382)
> events <- c(142892, 84876)
> prop.test(events, total, conf.level=0.90)


2-sample test for equality of proportions with continuity
  correction
data:  events out of total
X-squared = 3948.2, df = 1, p-value < 2.2e-16
alternative hypothesis: two.sided
90 percent confidence interval:
 0.02432700 0.02561555
sample estimates:
    prop 1     prop 2
0.10989976 0.08492849
```

Detecting Known Problems Using Patterns

- Patterns in the memory utilizations
  - Memory leak detection
- Patterns in the logs
  - Error keywords

Looking for known patterns: Deadlocks and memory leak

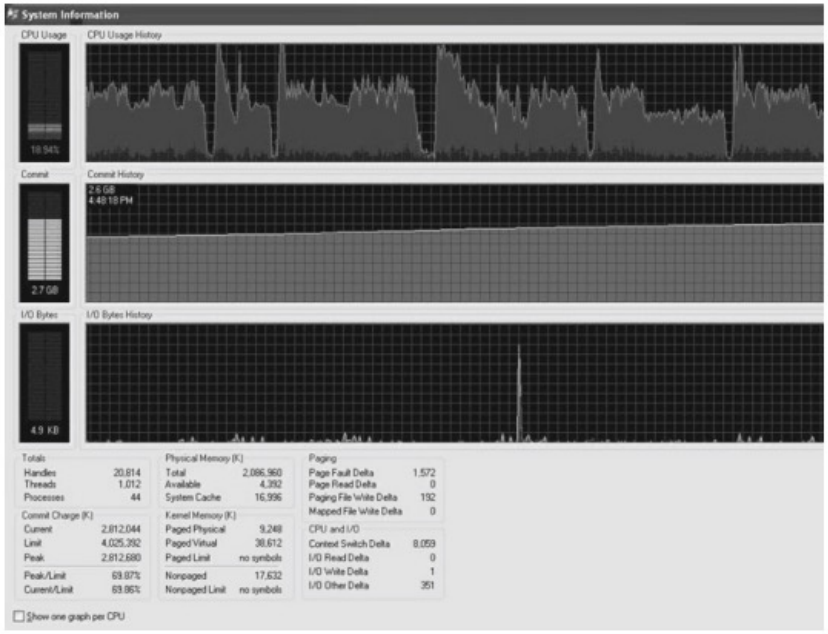
CPU

Memory

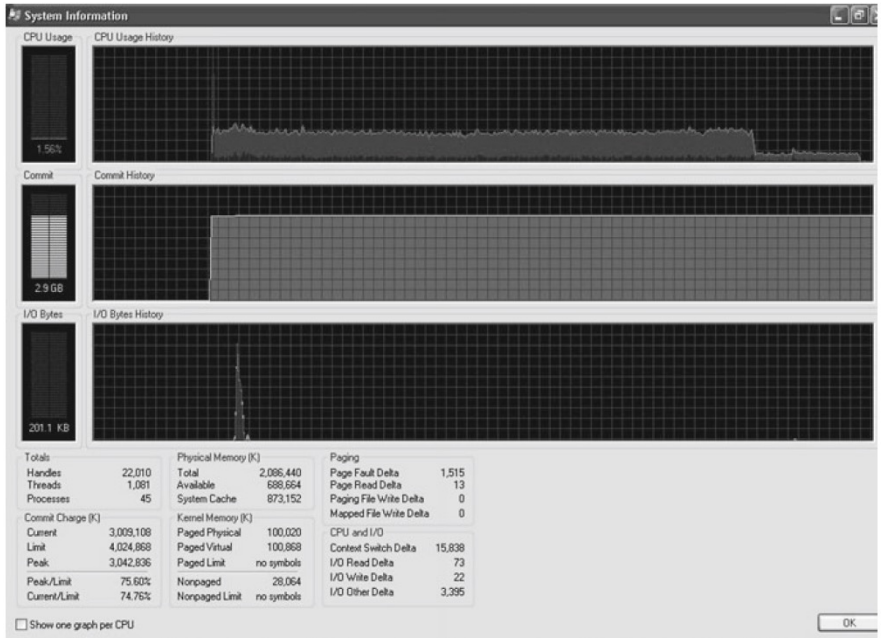
Performance data under **steady** load

[Avritzer et al., 2012]

Deadlocks and memory leak: before and after fix



Before fix



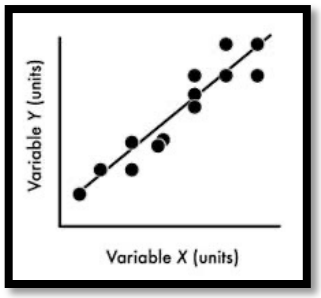
After fix

[Avritzer et al., 2012]

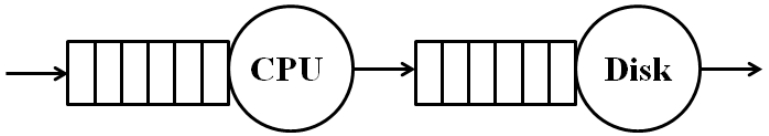
Tracking keywords in logs

ELK: https://www.elastic.co/what-is/elk-stack

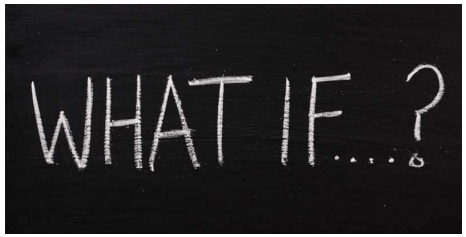
Building performance models

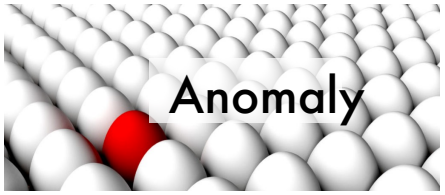


Statistical / data mining models (black box)



Queuing models (white box)





Profiling

Profiling

- A form of dynamic program analysis that measures the complexity of the program in terms of space (memory) or time, or the frequency and duration of function calls.
- Its objective is the **optimization of the program** and the management of resources.
- It is a process that helps to understand the behavior of a program.
- It also helps evaluate and compare performance of different architectures.
- Profiling has two important components: instrumentation and sampling.

Profiling: Instrumentation

- It is possible to collect data by external tools, but this data is not detailed enough and of a sufficient level of granularity.
- For this reason, instrumentation is used.
  - A technique that adds code (probes) in the monitored program to collect performance data.
- It is possible to add probes at several levels of the system.
  - Source code (manually or automatically)
  - Assisted by the compiler
  - Binary code

- Motivation for profiling:
  - Collect exactly the data needed and infer the locality of the data.
  - Control the granularity of data.
  - Control the measurement process by activating and deactivating probes.

Profiling: Instrumentation Design

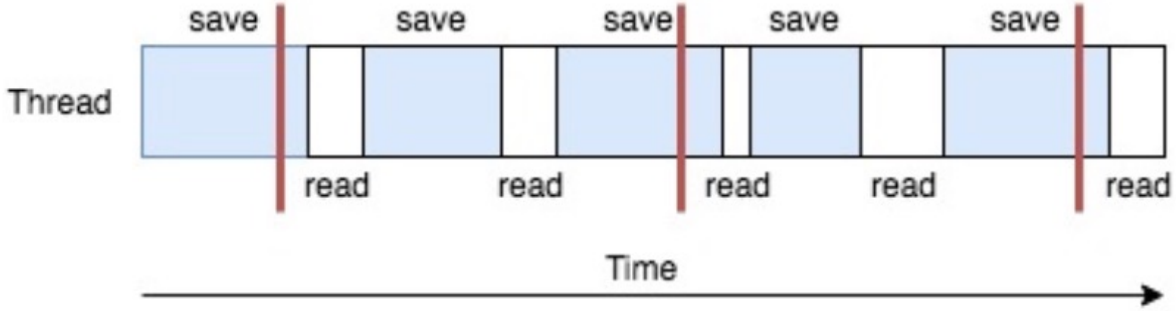
- Identify the events to be measured.
  - The events that are important for each scenario, including the start and end of key functions.
- Choose the level of granularity.
  - One could capture all the events but at a too high cost.
  - One could activate probes selectively at some points in the code and some components of the software.
  - One could activate some probes and then calculate the means, variances and distributions.
- Dynamically select the data to be saved.
  - Record data at runtime.
  - Use instrumentation parameters to vary metrics and their granularity.

Profiling: The pitfalls of instrumentation

- Instrumentation adds instructions at the start and end of an operation to count the operation execution time.
  - These instructions add overhead.
  - One could calculate the overhead and subtracts it from the runtime to make the measurement more precise.
- If the operation is too short, the overhead becomes considerable and the profiler cannot accurately compare the times between short operations and slow operations.
  - Then we can have a false positive: the profiler can identify a bottleneck that does not exist.
- Because instrumentation is an intrusive process, it is possible to identify "heisenbugs".
  - Bugs whose presence depends on the measurement process.
  - A phenomenon known as the "observer effect".

Profiling: Sampling

- Sampling does not affect the execution of the program.
  - No instruction is inserted in the source elbow nor in the compiled code.
- The operating system suspends the CPU at regular intervals and the profiler records the instruction that is currently executing.
- The profiler correlates the instruction with the corresponding point in the code.
- The profiler returns the frequency of execution of code points.
- Repeat profiling with sampling several times to obtain statistical significance.



Profiling : Sampling vs Instrumentation

- Sampling is less precise but much more efficient than instrumentation.
  - Sampling is based on approximations, so it requires several runs of profiling to converge its results.
  - Sampling is an external process of the application so it does not prevent software performance and it does not add any overhead (not exactly, why?).
- Sampling just captures a snapshot of the CPU, so it loses information.
  - We know which instruction is executing, but we do not know who called the instruction.
- If the profiled operation is too short (shorter than the sampling interval), the sampling will not capture it.
- If the operation or the profiled system is slow enough, instrumentation may be preferred.
  - Because the added overhead is insignificant compared to the execution time of operations.

Profiling: Automated Profiling

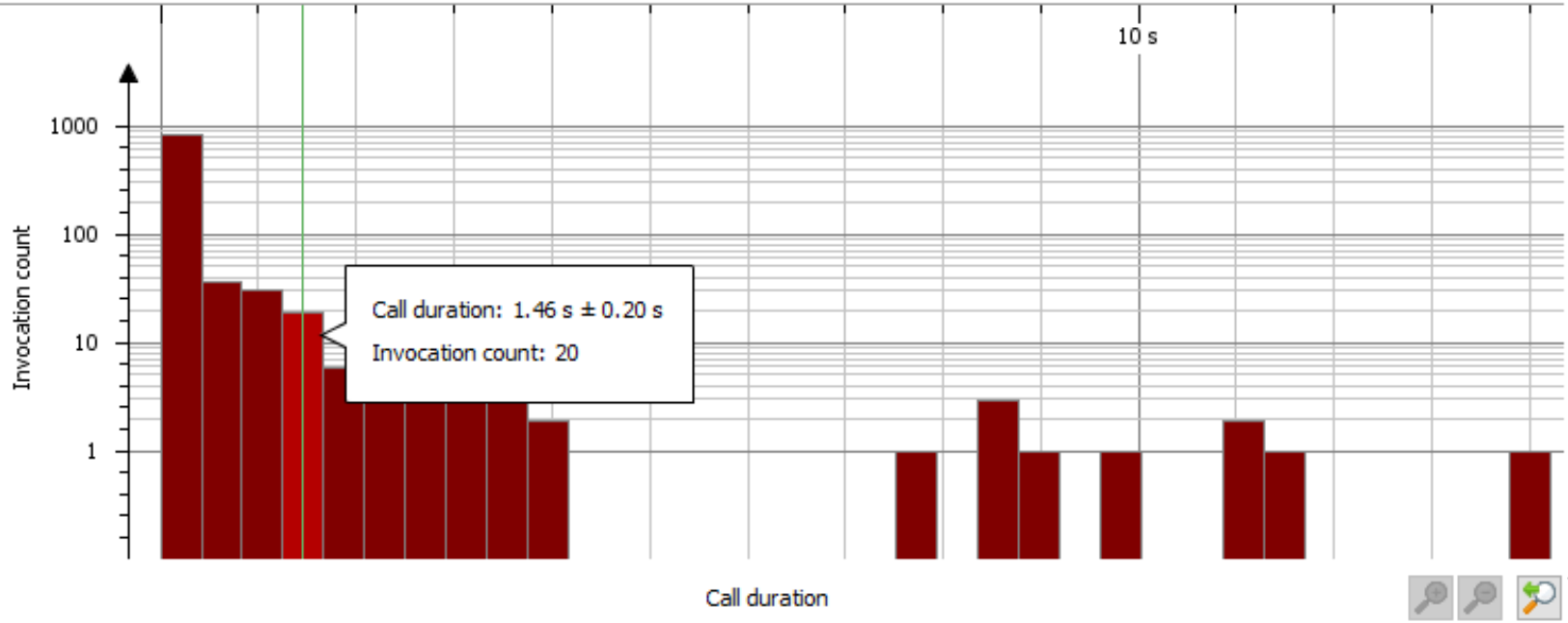
- Automated profiling facilitates optimization and guarantees continuous integration and continuous quality assurance.
- It also reduces optimization costs.
- Profiling tools are able to calculate a large number of measurements and produce detailed reports.
- Warning! Some profiling methods are characterized as intrusive, which can affect the results of the process.

JProfiler's CPU Profiling

Thread status: All states

Method	Total Time	Inv.	Avg. Time	Median Time	Min. Time	Max. Time	Std. Dev.	Outlier Coeff.
weblogic.work.ExecuteThread.waitForR...	487 s	1,005	485 ms	51 µs	51 µs	14,295 ms	1,200 ms	280,305.16
weblogic.invocation.ComponentInvocati...	155 s	1,943	79,850 µs	76 µs	76 µs	26,472 ms	1,140 ms	348,320.13
weblogic.work.ExecuteThread.execute(...	153 s	1,188	129 ms	102 µs	102 µs	26,472 ms	1,456 ms	259,532.71
weblogic.work.SelfTuningWorkManagerI...	153 s	1,188	129 ms	93 µs	93 µs	26,472 ms	1,456 ms	284,648.74
weblogic.work.PartitionUtility.runWorkU...	153 s	1,188	129 ms	89 µs	89 µs	26,472 ms	1,456 ms	297,441.93
weblogic.work.LivePartitionUtility.doRun...	153 s	1,188	129 ms	87 µs	87 µs	26,472 ms	1,456 ms	304,279.68
weblogic.invocation.ComponentInvocati...	153 s	1,187	129 ms	86 µs	86 µs	26,472 ms	1,457 ms	307,817.80
weblogic.work.SelfTuningWorkManagerI...	151 s	704	215 ms	32 µs	32 µs	26,472 ms	1,887 ms	827,259.00
weblogic.timers.internal.TimerThread.ac...	147 s	1,083	136 ms	1 µs	1 µs	4,218 ms	243 ms	4,218,763.00

Class View Filters



Profiling vs Performance testing: when to use them in a project?

Profiling → Performance testing

- We can use profiling to understand the behavior of our program ...
- ... and identify the use of resources (CPU, memory etc.)
- After that, we can define the thresholds and objectives for the performance and test them.
- We can also train or provide inputs for our performance models.

Performance testing → Profiling

- Testing will indicate the presence of performance issues.
- According to this indication, profiling will reveal the exact point of the bottleneck.
- After, we optimize the code and rerun the tests.

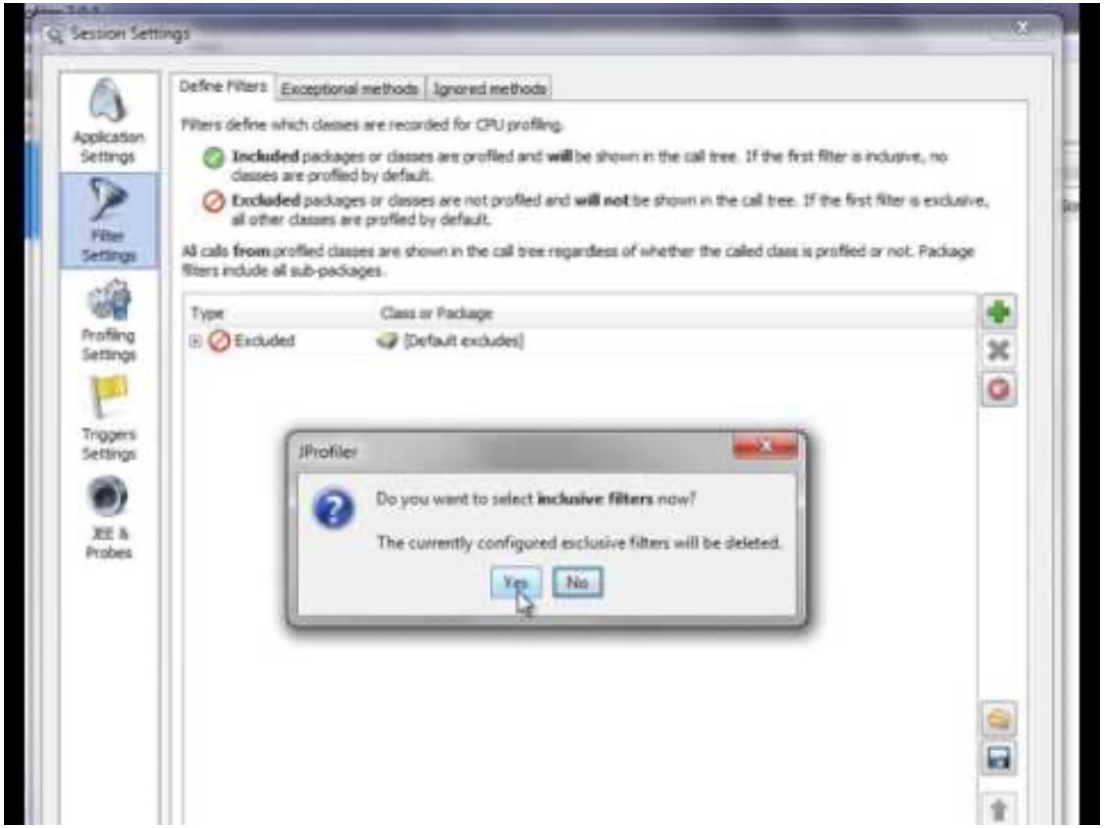
Exercise 1: Use JMeter to perform performance testing

- Follow the tutorial to load-test a demo website (https://blazedemo.com) using JMeter:
  - Tutorial: https://www.blazemeter.com/blog/getting-started-jmeter-basic-tutorial
  - JMeter: https://jmeter.apache.org

- You may also consider other systems:
  - Some examples: https://www.quora.com/Which-sample-website-I-can-use-to-test-using-JMeter

Exercise 2: Sampling and instrumentation using JProfiler

- Use the instrumentation and sampling methods to profile an application (e.g., your IDE) running on your local machine.
- Tutorial: https://youtu.be/XMUNKBxdQYk



Exercise 3: Design realistic loads for performance testing

- Based on the following sample logs, design a use-case model using the Markov chain

#	Log Lines
1	time=1, thread=1, session=1, receiving new user registration request
2	time=1, thread=1, session=1, inserting user information to the database
3	time=1, thread=2, session=2, user=Jack, browse catalog=novels
4	time=1, thread=2, session=2, user=Jack, sending search queries to the database
5	time=3, thread=1, session=1, user=Tom, registration completed, sending confirmation email to the user
6	time=3, thread=2, session=2, database connection error: session timeout
7	time=4, thread=1, session=1, fail to send the confirmation email, number of retry = 1
8	time=6, thread=2, session=2, user=Jack, successfully retrieved data from the database
9	time=7, thread=2, system health check
10	time=8, thread=1, session=1, registration email sent successfully to user=Tom
11	time=9, thread=2, session=3, user=Tom, browse catalog=travel
12	time=10, thread=2, session=3, user=Tom, sending search queries to the database
13	time=10, thread=3, session=4, user=Jim, updating user profile
14	time=11, thread=3, session=4, user=Jim, database error: deadlock

TP2 - Performance Efficiency

- **Performance/load testing**
- **Performance Profiling**
- **Due on November 3rd**