Software Performance

Sarah Nadi
nadi@ualberta.ca
Department of Computing Science
University of Alberta

CMPUT 402— Software Quality

Learning Goals

- Learn about different types of performance testing (& their goals)
- Learn about metrics calculated during performance testing
- Get exposed to performance testing tools
- Learn about tools/techniques for capturing performance problems early

What is good vs. bad performance?

- “A well-performing application is one that lets the end user carry out a given task without undue *perceived* delay or irritation”

- no blank screen during login

- browsing a website and placing an order without delay or frustration

- …

Molyneaux, Ian. *The art of application performance testing: from strategy to tools*. " O'Reilly Media, Inc.", 2014.

Performance Measurement

- Key Performance Indicators (KPIs)

Service-oriented indicators:

- **Availability**
- **Response time**

Efficiency-oriented indicators:

- **Throughput**
- **Capacity utilization**

Molyneaux, Ian. *The art of application performance testing: from strategy to tools*. " O'Reilly Media, Inc.", 2014.

Availability

- The amount of time an application is available to the end user
- From a performance perspective: the user’s inability to make effective use of the application because the application is not responding or response time is significantly downgraded means that the application is not really available to the user.

Molyneaux, Ian. *The art of application performance testing: from strategy to tools*. " O'Reilly Media, Inc.", 2014.

Response Time

- The amount of time it takes for the application to respond to a user request
- <= 0.1s for seamless manipulation; <=1s for feeling of free navigation; <=10s before users switch/interrupt the operation (https://www.nngroup.com/articles/response-times-3-important-limits/)

Molyneaux, Ian. *The art of application performance testing: from strategy to tools*. " O'Reilly Media, Inc.", 2014.

Throughput

- Number of application-oriented events the software can process (within a given time)

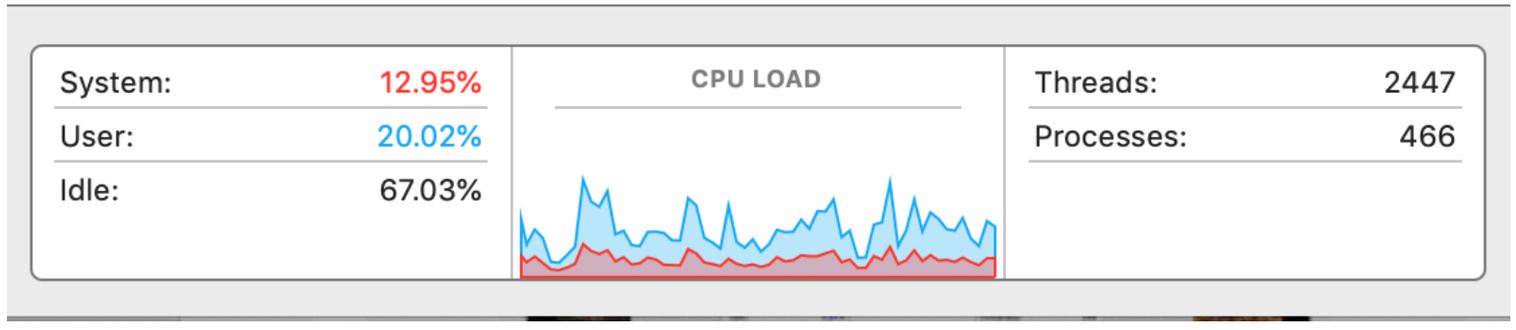
- E.g., number of requests handled by a web page within a given period of time

Molyneaux, Ian. *The art of application performance testing: from strategy to tools*. " O'Reilly Media, Inc.", 2014.

Capacity Utilization

- The percentage of the capacity of a resource that is being used

- E.g., how much network bandwidth is being consumed by application traffic or the amount of memory used on a web server farm when 1,000 visitors are active.



Performance Testing

- Goal is to measure the quality (i.e., non-functional) attributes of the system related to its performance

- Typically measuring things like response time, utilization, and throughput

System-level Performance Testing Techniques

- Baseline testing
- Load testing
- Soak testing
- Stress testing

Baseline Testing

Baseline Testing

Goal:
Establish a point of comparison for further tests

- Execute a single transaction as a single user for a set period of time or for a set number of transaction interactions

- Carried out under normal conditions

Load Testing

Load Testing

Goal:
Understand the behavior of the system under a normal/expected load

- Load testing answers the following (example) questions:

➡ What is the maximum load the application is able to hold before behaving unexpectedly?

➡ How much data is the Database able to handle before observing delays or crashes?

➡ Are there network-related issues to be addressed? Is the current infrastructure sufficient?

Soak (Stability) Testing

Soak (Stability) Testing

Goal:
Identify performance problems that appear over extended periods of time

- Supply expected load to the application continuously over a period of time

- Can catch problems that occur only after the system is running for a while (E.g., memory leaks)

Stress Testing

Stress Testing

Goal:
See how the application performs under unfavorable conditions & ensure it fails and recovers gracefully

- How?
  - Overwhelm the system resources with requests
  - Insert large volume of data in DB
  - Start intensive process (e.g., virus scanner) on all machines
- It is important that the system is able to recover from the failure

Spike Testing

Spike Testing

Goal:
Make sure the system can handle “bursts” of higher user or system activity

- Similar to stress test but focuses on high load for short duration (i.e., use a sudden ramp-up in number of users)

Performance Testing Process

1. Decide on the testing environment
2. Identify performance metrics
3. Plan and design performance test (take into account diff. user types, data, scenarios etc)
4. Configure the test environment
5. Implement the tests
6. Execute tests
7. Analyze, report, retest

https://stackify.com/ultimate-guide-performance-testing-and-software-testing/

1. Decide on the Testing Environment

- What kind of environment (can you afford)?
  - subset of production system w/ lower-spec servers
  - subset of production w/ few spec servers
  - replica of production
  - actual production
- Obviously, the closer to the production environment, the better but there are usually resource constraints

2. Identify Performance Metrics

- **Response time:** total time to send a request and get a response
- **Latency:** how long it takes to receive the first byte after a request is sent
- **Error rate:** percentage of requests resulting in errors
- **Concurrent users:** how many active users at any point in time (most common measure of load)
- **CPU utilization**
- **Memory utilization**
- **…**

3. Plan and Design Performance Tests

- Which use cases do you want to test (different types of users, different scenarios etc)?

- Which checkpoints do you care about?

3. Plan and Design Performance Tests

- Which use cases do you want to test (different types of users, different scenarios etc)?

- Which checkpoints do you care about?

```
Example use case:
make a purchase order
as a logged in user
```

```
Example checkpoints:
- log into website
- click checkout
- process payment
  information
- show final
  confirmation
```

4. Configure the Test Environment

- Ideally, automate the creation and tear down of the testing environment to allow repeatability
  - E.g., Chef, Ansible, Spinnaker, Heroku
- Set up the tools you will use for monitoring

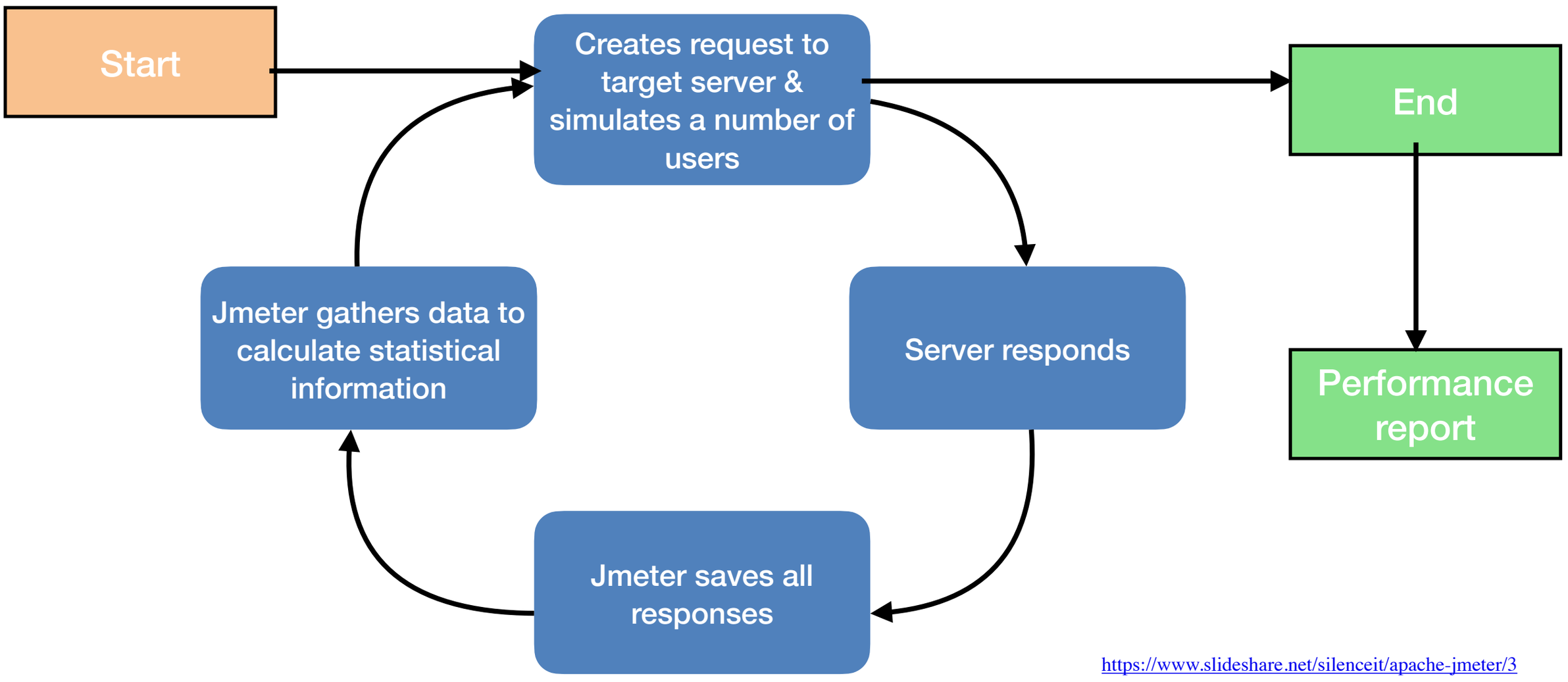
Performance Testing & Monitoring Tools

Apache JMeter

- Open-source tool https://jmeter.apache.org/
- Provides load and performance tests and produces HTML reports
- Includes a Test IDE for Test Plan recording (from browser or native applications)

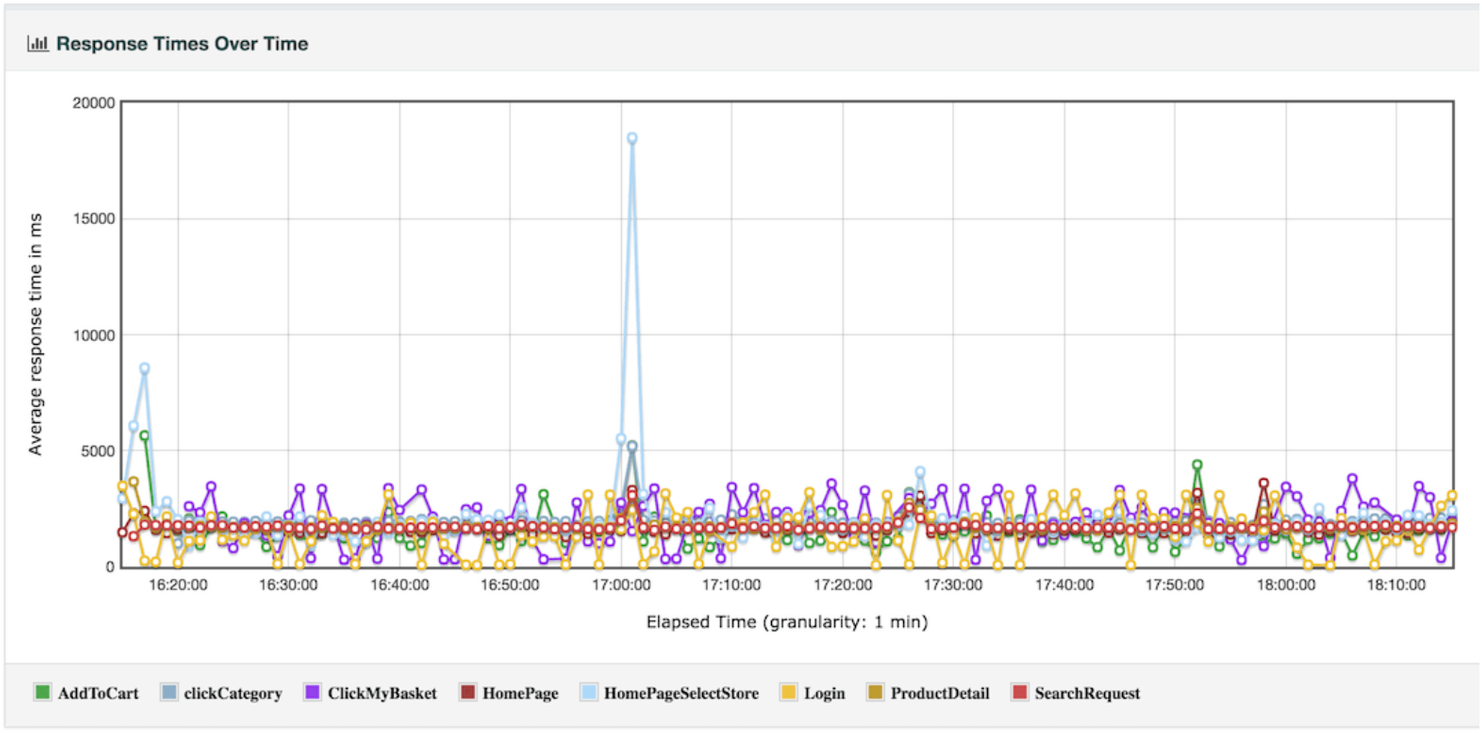
How Does JMeter Work?

Jmeter simulates a group of users sending requests to a target server, and returns statistics that show the performance of the target server/application through graphical diagrams



https://www.slideshare.net/silenceit/apache-jmeter/3

Apache JMeter



Statistics

Requests	Executions			Response Times (ms)							Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Throughput	Received	Sent
Total	**750**	**0**	**0.00%**	**3.49**	**0**	**67**	**5.00**	**6.00**	**13.00**	**5.14**	**14.69**	**0.00**
SC01_00_HomePage	250	0	0.00%	2.77	0	67	4.00	5.00	10.92	1.79	12.31	0.00
SC01_01_Forms	250	0	0.00%	3.79	1	53	5.00	6.00	13.98	1.79	1.52	0.00
SC01_02_SendForms	250	0	0.00%	3.91	1	17	5.00	6.45	12.49	1.77	1.49	0.00

Covered in lab next week!

https://www.youtube.com/watch?v=l4nxzw6PYg4

5. Implement the Tests

- Create/implement your test plans in the target tool

6. Execute the Tests

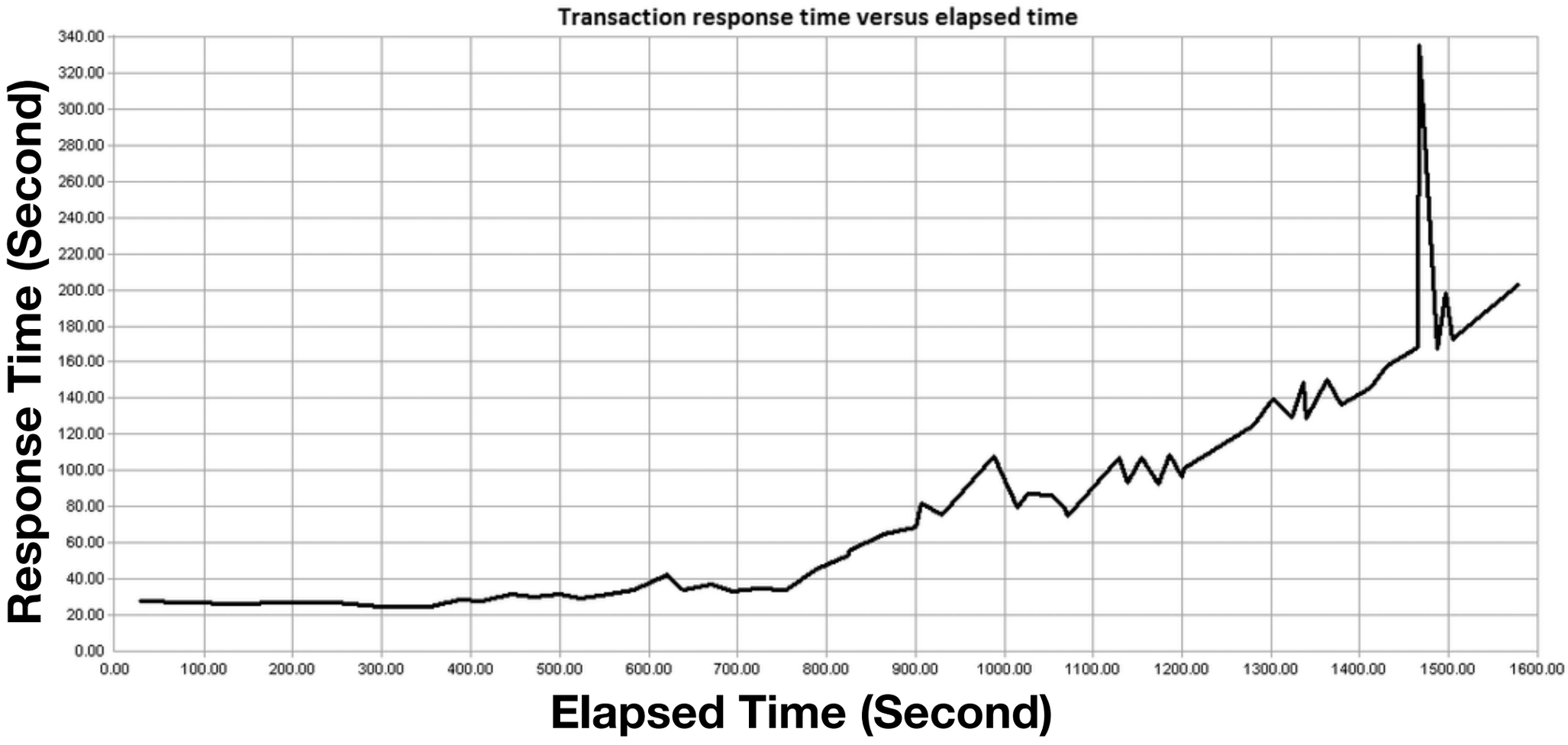
- Run the actual tests!

7. Analyze, Report, Repeat

- Performance testing doesn’t have pas/fail results as functional tests do

7. Analyze, Report, Repeat

- Performance testing doesn’t have pas/fail results as functional tests do

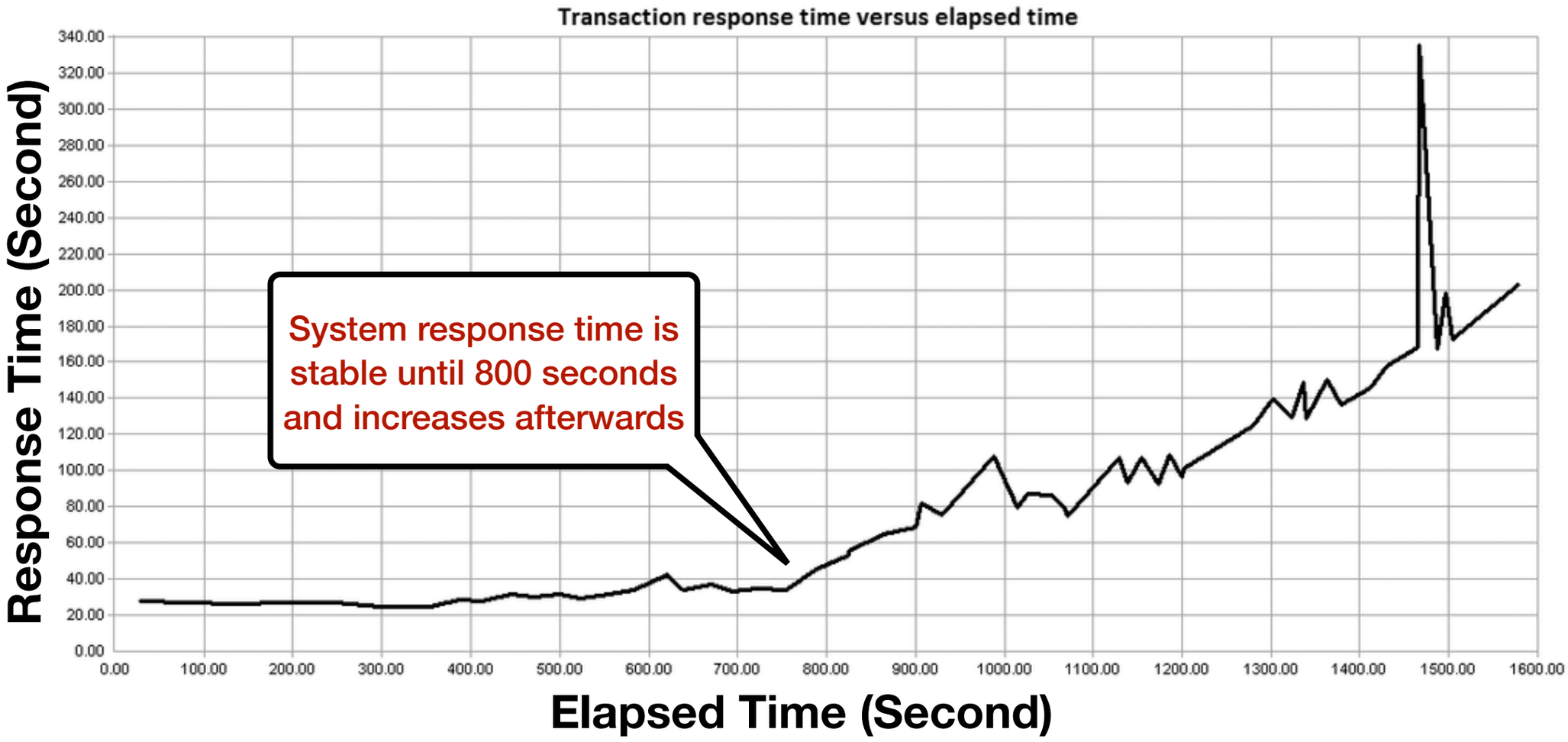


Molyneaux. The Art of Application Performance Testing. 2009

What can we conclude?

7. Analyze, Report, Repeat

- Performance testing doesn’t have pas/fail results as functional tests do

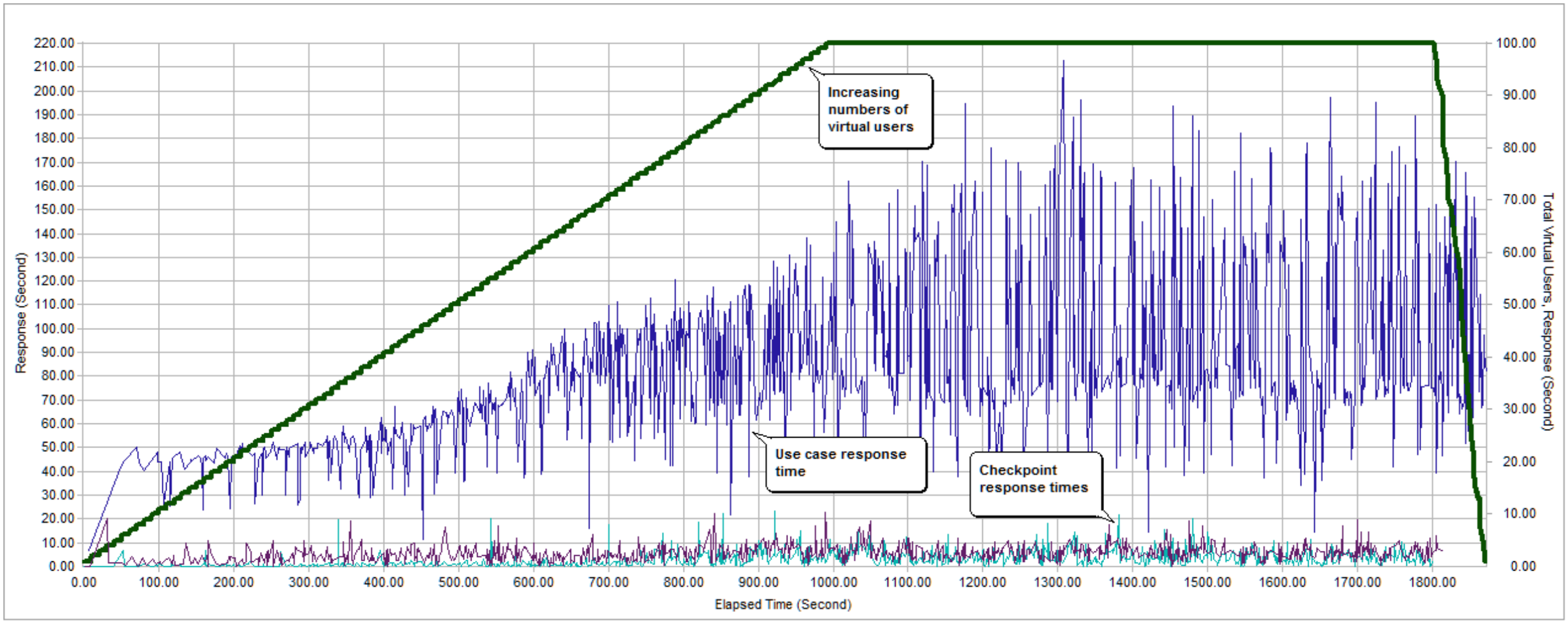


Molyneaux. The Art of Application Performance Testing. 2009

What can we conclude?

7. Analyze, Report, Repeat

- Performance testing doesn’t have pas/fail results as functional tests do



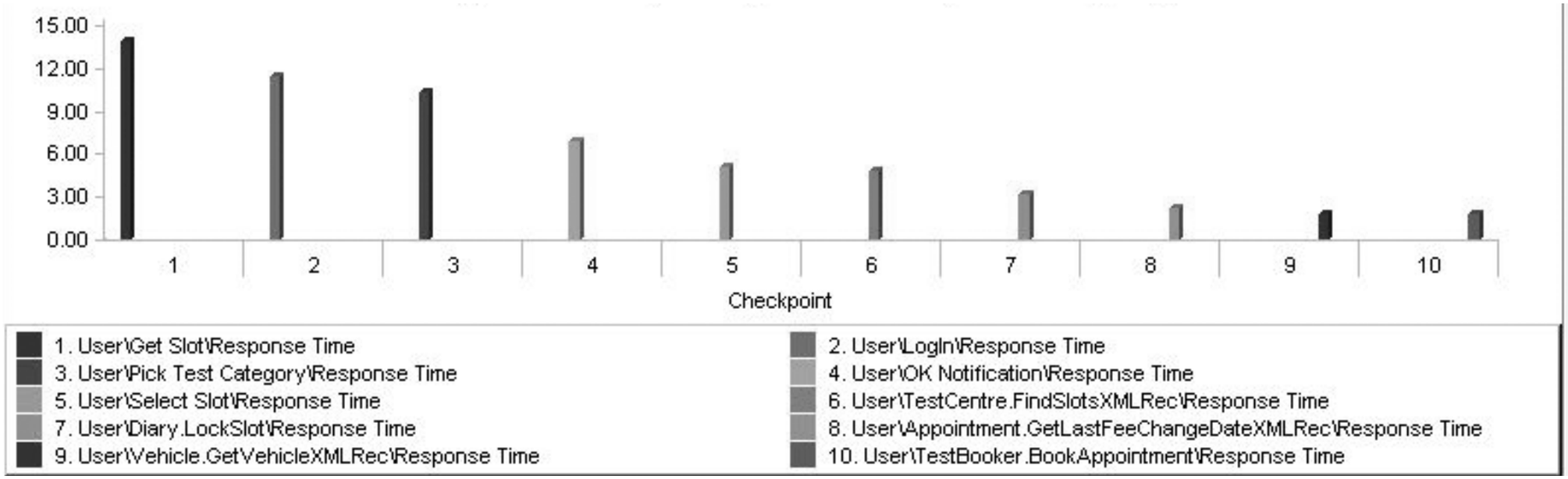
Molyneaux, Ian. *The art of application performance testing: from strategy to tools*. " O'Reilly Media, Inc.", 2014.

7. Analyze, Report, Repeat

- Performance testing doesn’t have pas/fail results as functional tests

Name	Min	Mean	Max	Last	Std.Dev.
Transaction Response Time	24.9380	86.0258	335.1410	202.7350	61.5771
Total Running Virtual Users	10.0000	458.7383	913.0000	913.0000	261.8142
Login Response Time	0.7660	7.3413	24.2350	7.1870	5.3617
Security Question Response Time	0.5930	6.7524	20.6250	6.9690	5.0116
Click Custom Statements Response Time	0.8120	6.0654	35.0470	4.7970	5.2828
Show Statement Response Time	0.7970	6.0582	15.9690	8.4840	4.0250
Logoff Response Time	1.2030	15.1568	246.0940	10.7820	32.2105

10 worst Performing Checkpoints



Other Performance Testing/ Assurance Activities

- Dynamic Analysis (Instrumentation/profiling)
- Code review & code analysis tools

Molyneaux, Ian. *The art of application performance testing: from strategy to tools*. " O'Reilly Media, Inc.", 2014.

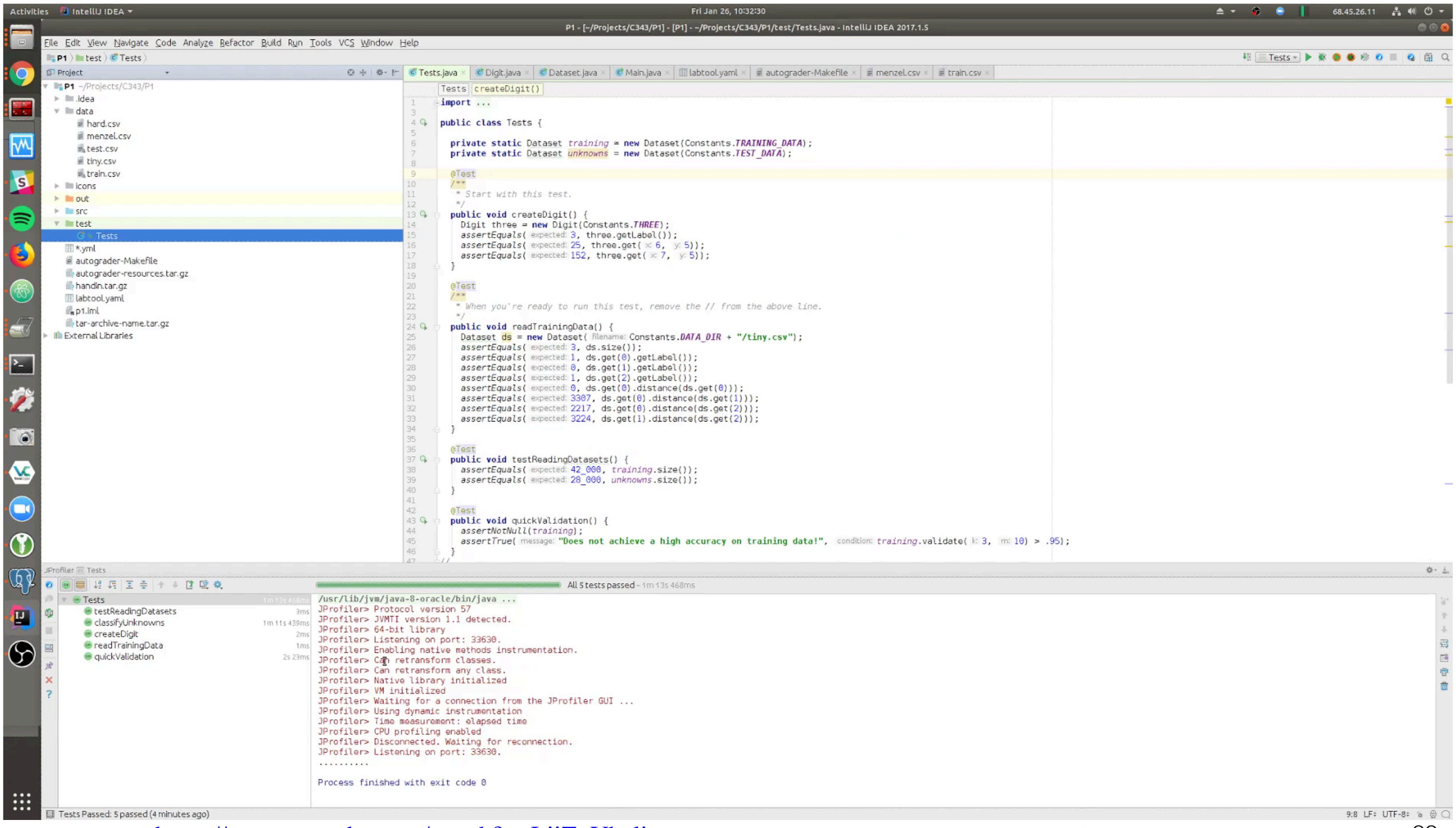
Profiling

- Use tools that instrument your program (i.e., add extra things in your code that will trigger fact recording at run time) to record information about performance, method call, bottlenecks etc.

- Can be used to profile your existing unit or integration tests to already discover potential problems before explicit performance testing

- Can potentially be integrated into your continuous integration process (but note that profiling does slow down test execution) — could potentially profile only critical components

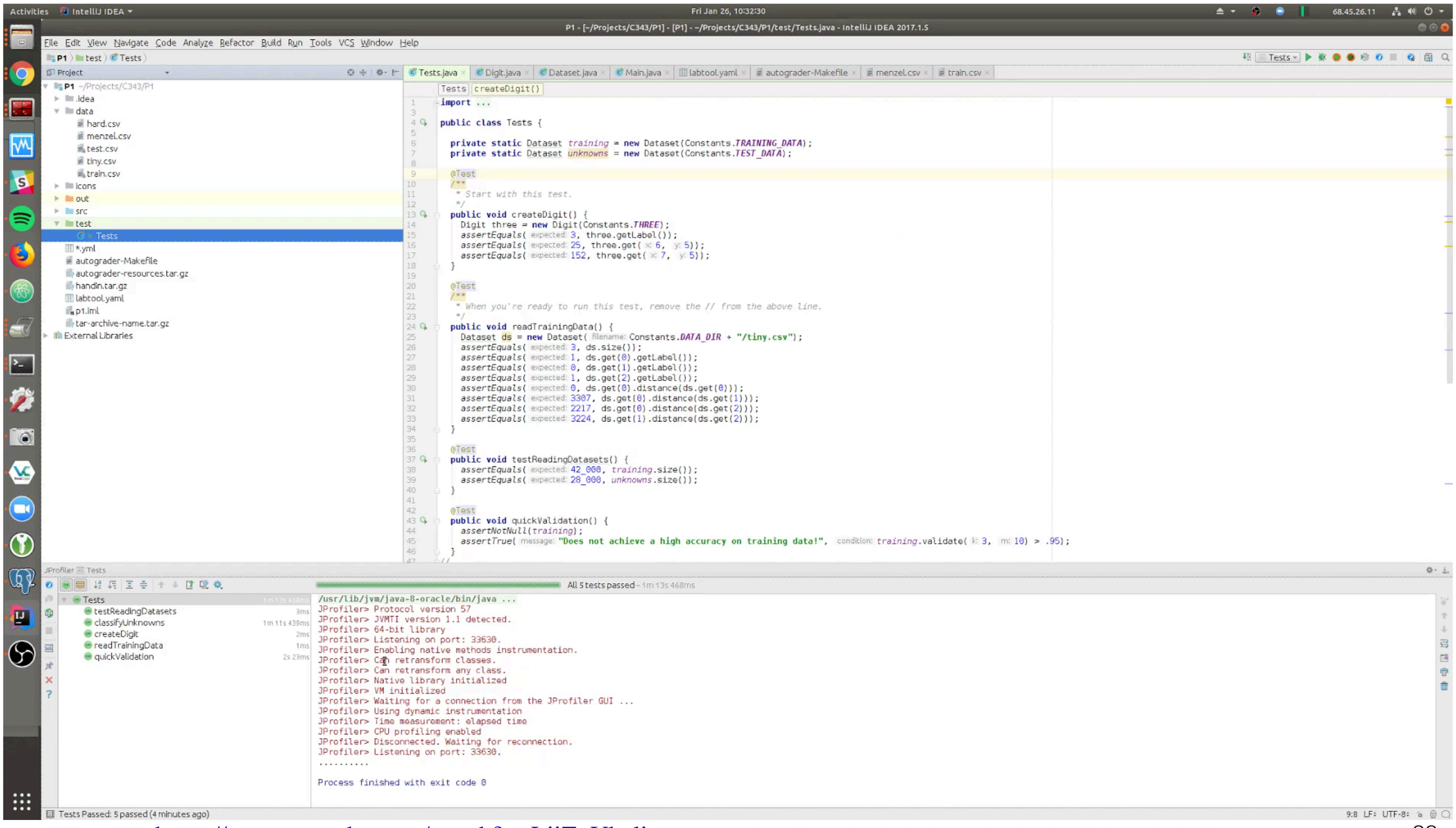
Example Tool: JProfiler



https://www.youtube.com/watch?v=LiiZnYbcliw

https://www.ej-technologies.com/products/jprofiler/overview.html

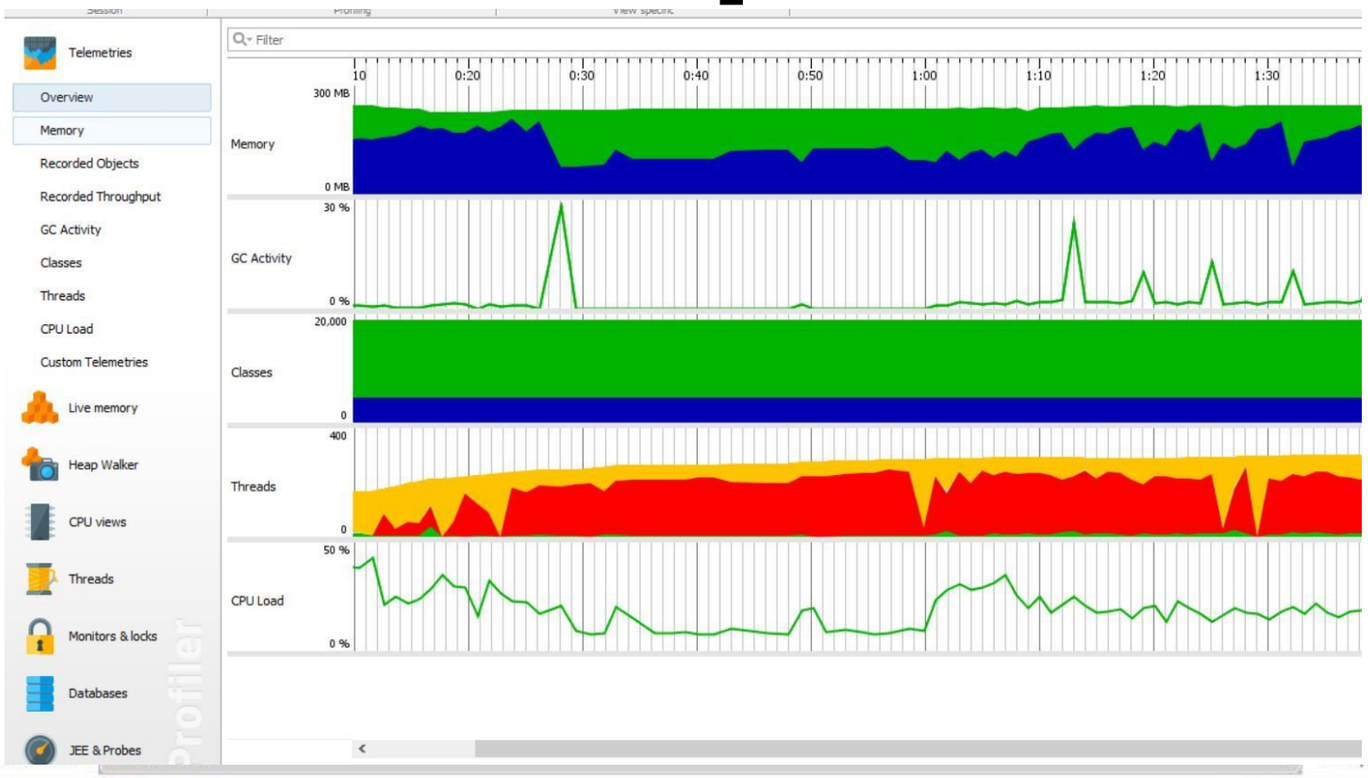
Example Tool: JProfiler

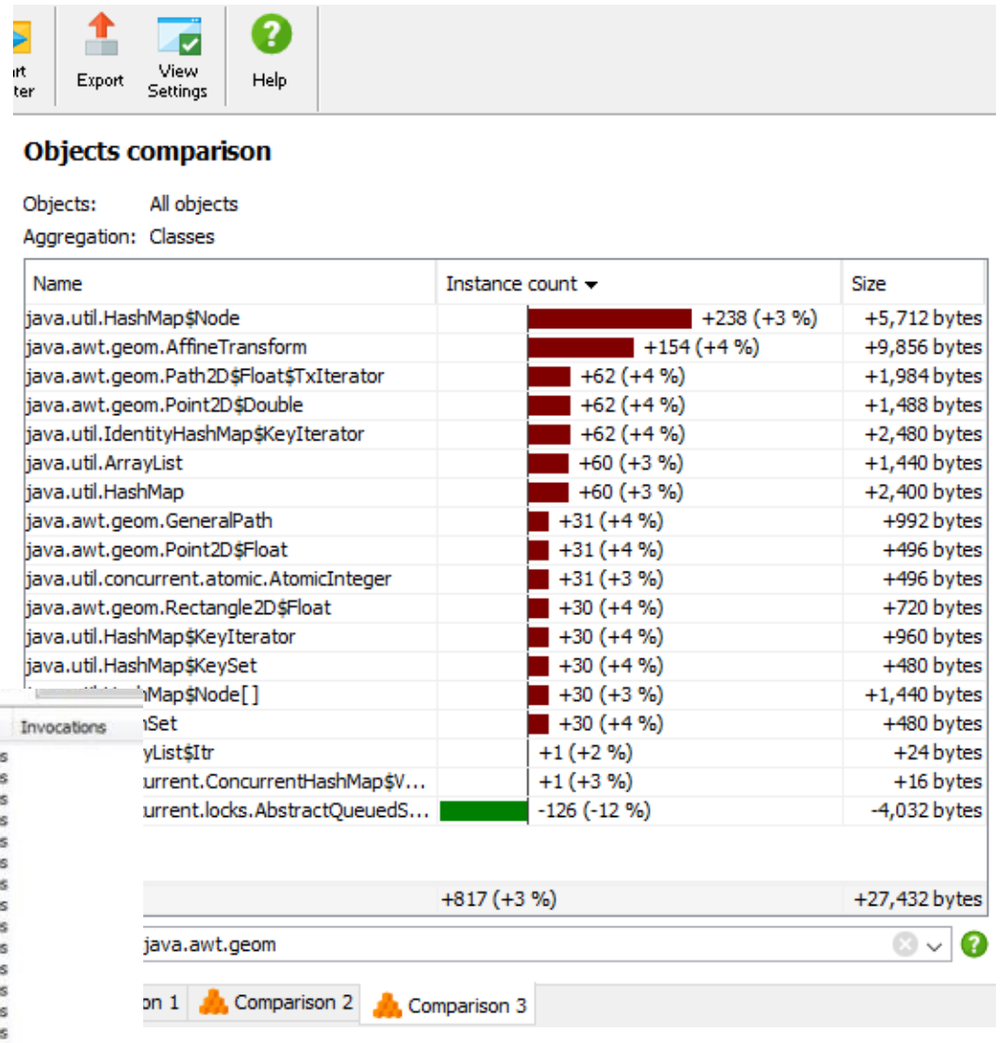


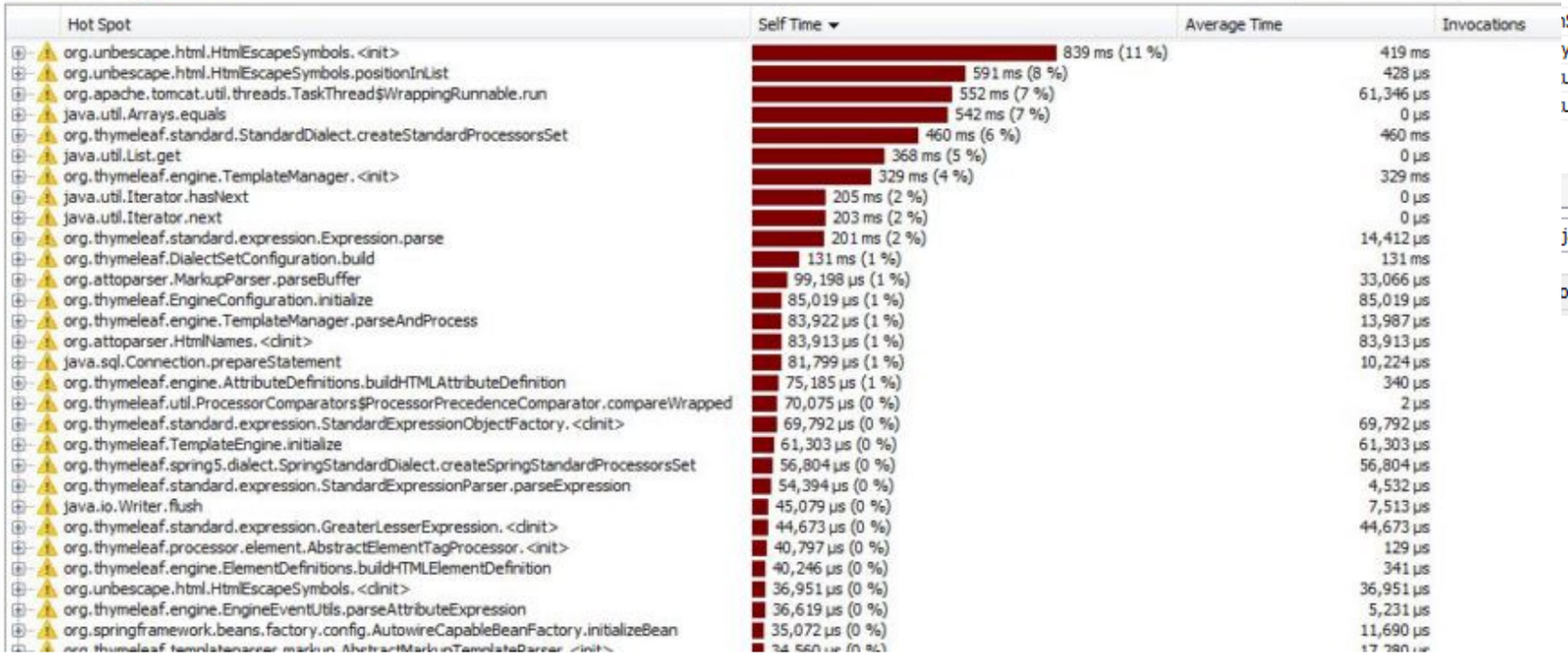
https://www.youtube.com/watch?v=LiiZnYbcliw

https://www.ej-technologies.com/products/jprofiler/overview.html

Example Tool: JProfiler







Performance Code Review

- Example questions/things to look for:
  - ➡ Calls to external services needed?
  - ➡ Inefficient coding patterns?
  - ➡ Inefficient data structures?
  - ➡ Unnecessary locking of critical resources?
  - ➡ Release of memory and resources?
  - ➡ Efficient caching used?
  - ➡ Reflection, timeouts, …?
  - ➡ Parallelism potential vs. overhead?

https://blog.jetbrains.com/upsource/2015/08/06/what-to-look-for-in-a-code-review-performance/

Inefficient Coding Patterns (DB Example)

CustomerDao.java src/main/java/com/mechanitis/blog/upsource/customer +20 Side-by-side diff | View file

```
11 11       private final Database database = new Database();
12 12
   13 +     public List<Customer> getAllCustomers() {
   14 +         List<Integer> customerIds = getAllCustomerIds();
   15 +         ArrayList<Customer> customers = new ArrayList<>();
   16 +         try (Connection connection = database.getConnection();
   17 +              Statement statement = connection.createStatement()) {
   18 +             for (Integer customerId : customerIds) {
   19 +                 try (ResultSet rs = statement.executeQuery("SELECT * FROM Customers WHERE id = " + customerId)) {
   20 +                     rs.next();
   21 +                     customers.add(new Customer(
   22 +                             customerId,
   23 +                             rs.getString("first"),
   24 +                             rs.getString("last")
   25 +                     ));
   26 +                 }
   27 +             }
   28 +         } catch (SQLException e) {
   29 +             doDatabaseErrorHandling(e);
   30 +         }
   31 +         return customers;
   32 +     }
13 33
14 34       public List<Integer> getAllCustomerIds() {
```

Inefficient Coding Patterns (DB Example)

CustomerDao.java src/main/java/com/mechanitis/blog/upsource/customer +20 Side-by-side diff | View file

```
11 11      private final Database database = new Database();
12 12
   13 +    public List<Customer> getAllCustomers() {
   14 +        List<Integer> customerIds = getAllCustomerIds();
   15 +        ArrayList<Customer> customers = new ArrayList<>();
   16 +        try (Connection connection = database.getConnection();
   17 +             Statement statement = connection.createStatement()) {
   18 +            for (Integer customerId : customerIds) {
   19 +                try (ResultSet rs = statement.executeQuery("SELECT * FROM Customers WHERE id = " + customerId)) {
   20 +                    rs.next();
   21 +                    customers.add(new Customer(
   22 +                            customerId,
   23 +                            rs.getString("first"),
   24 +                            rs.getString("last")
   25 +                    ));
   26 +                }
   27 +            }
   28 +        } catch (SQLException e) {
   29 +            doDatabaseErrorHandling(e);
   30 +        }
   31 +        return customers;
   32 +    }
13 33
14 34      public List<Integer> getAllCustomerIds() {
```

Repeated calls to a DB inside a loop… needed?

Parallelism Potential vs. Overhead

MoodAnalyser.java src/main/java/com/mechanitis/blog/upsource/social +1 −5 Side-by-side diff | View file

```
 3  3  import java.util.HashMap;
13  8  public class MoodAnalyser {

35 30      public static String analyseMood(String tweet) {
36 31          return Stream.of(tweet.split("\\s+"))
   32  +                    .parallel()
37 33                      .map(String::toLowerCase)
38 34                      .map(WORD_TO_MOOD::get)
```

https://blog.jetbrains.com/upsource/2015/08/06/what-to-look-for-in-a-code-review-performance/

Parallelism Potential vs. Overhead

MoodAnalyser.java src/main/java/com/mechanitis/blog/upsource/social +1 −5 Side-by-side diff | View file

```
 3  3  import java.util.HashMap;
13  8  public class MoodAnalyser {

35 30      public static String analyseMood(String tweet) {
36 31          return Stream.of(tweet.split("\\s+"))
   32  +                    .parallel()
37 33                      .map(String::toLowerCase)
38 34                      .map(WORD_TO_MOOD::get)
```

A single tweet is 140 characters so not that many words.. overhead of splitting execution is likely bigger than any performance gain

https://blog.jetbrains.com/upsource/2015/08/06/what-to-look-for-in-a-code-review-performance/

Tooling to Catch Bad Performance Patterns

- SpotBugs Performance Patterns:
  - “SBSC: Method concatenates strings using + in a loop”
  - “Dm: Method invokes inefficient new String(String) constructor”
  - “Dm: Method invokes inefficient new String() constructor”
  - "Dm: Method invokes toString() method on a String”
  - “Dm: Explicit garbage collection; extremely dubious except in benchmarking code”
  - “Dm: Method invokes inefficient Boolean constructor; use Boolean.valueOf() instead”
  - “Bx: Method invokes inefficient Number constructor; use static valueOf instead"
  - “Bx: Method invokes inefficient floating-point Number constructor; use static valueOf instead”
  - …

https://spotbugs.readthedocs.io/en/stable/bugDescriptions.html#performance-performance

Extra Resources

- Performance testing tools: https://techcommunity.microsoft.com/t5/testingspot-blog/what-are-the-best-performance-testing-tools/ba-p/367774#

- How YouTube does performance testing (5min video): https://developers.google.com/web/shows/google-io/2014/web-performance-testing-at-youtube

- Find performance problems in your own webpage: https://developers.google.com/speed/pagespeed

- Performance related tools for Google Chrome: https://developers.google.com/web/fundamentals/performance/get-started/measuringperf-2

Summary

- You can still think of performance during regular development activities (e.g., performance code review, profiling tests)

- Performing various types of performance testing is essential for understanding how your system behaves

- There are various performance testing tools that can help in the process