Context-Driven Performance Testing

Alexander Podelko
alex.podelko@oracle.com
alexanderpodelko.com/blog
@apodelko

February 20, 2019

About the Speaker

- **Alexander Podelko**
- **Specializing in performance since 1997**
- **Currently Consulting Member of Technical Staff at Oracle (Stamford, CT office)**
- **Performance testing and optimization of Enterprise Performance Management (EPM) a.k.a. Hyperion products**
- **Board director at Computer Measurement Group (CMG) – non-profit organization of performance and capacity professionals**

Agenda

- *Context-Driven Testing*
- Early Performance Testing
  - Exploratory, Continuous
- Environment / Scope / Granularity
- Load generation
- Performance Testing / Engineering Strategy

3

Context-Driven Testing

- Context-driven approach was initially introduced by James Bach, Brian Marick, Bret Pettichord, and Cem Kaner
  - http://context-driven-testing.com
- Declared a “school” in 2001 (Lessons Learned in Software Testing)
  - Became political

4

Basic Principles

- The value of any practice depends on its context.
- There are good practices in context, but there are no best practices.
- People, working together, are the most important part of any project's context.
- Projects unfold over time in ways that are often not predictable.

5

Basic Principles

- The product is a solution. If the problem isn't solved, the product doesn't work.
- Good software testing is a challenging intellectual process.
- Only through judgment and skill, exercised cooperatively throughout the entire project, are we able to do the right things at the right times to effectively test our products.

6

"Traditional" Approach

- Load / Performance Testing is:
  - Last moment before deployment
    - Last step in the waterfall process
    - Checking against given requirements / SLAs
    - Throwing it back over the wall if reqs are not met
  - System-level
  - Realistic workload
    - With variations when needed: stress, uptime, etc.
  - Lab environment
    - Often scale-down
  - Protocol level record-and-playback
    - Expensive tools requiring special skills

7

Agenda

- Context-Driven Testing
- ***Early Performance Testing***
  - ***Exploratory, Continuous***
- Environment / Scope / Granularity
- Load generation
- Performance Testing / Engineering Strategy

8

Agile Development

- Agile development should be rather a trivial case for performance testing
  - You have a working system each iteration to test early by definition.
  - You need performance engineer for the whole project
    - Savings come from detecting problems early
- You need to adjust requirements for implemented functionality
  - Additional functionality will impact performance

9

The Main Issue on the Agile Side

- It doesn't [always] work this way in practice
- That is why you have "Hardening Iterations", "Technical Debt" and similar notions
- Same old problem: functionality gets priority over performance

10

The Main Issue on the Testing Side

- Performance Engineering teams don't scale well
  - Even assuming that they are competent and effective
- Increased volume exposes the problem
  - Early testing
  - Each iteration
- Remedies: automation, making performance everyone's job

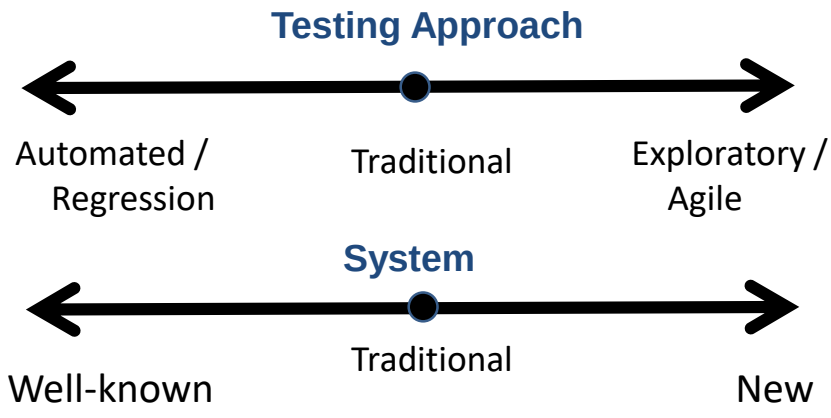
11

Early Testing - Mentality Change

- Making performance everyone's job
- Late record/playback performance testing -> Early Performance Engineering
- System-level requirements -> Component-level requirements
- Record/playback approach -> Programming to generate load/create stubs
- "Black Box" -> "Grey Box"

12

Continuum of Options



13

Exploratory Performance Testing

- Rather alien for performance testing, but probably more relevant than for functional testing
- We learn about system's performance as we start to run test
  - Only guesses for new systems
- Rather a performance engineering process bringing the system to the proper state than just testing

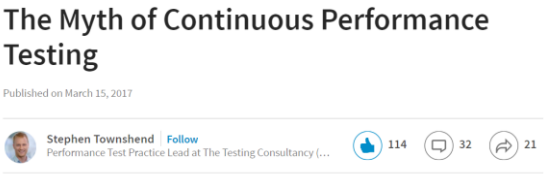
14

Continuous Performance Testing

- You see Performance CI presentations at every conference nowadays….

and

- Still opinions vary
  - From “full automation” to:


The Myth of Continuous Performance Testing

Published on March 15, 2017

Stephen Townshend | Follow
Performance Test Practice Lead at The Testing Consultancy (…

114 32 21

15

Different Perspectives

- Consultant: need to test the system
  - In its current state
  - Why bother about automation?
  - External or internal
- Performance Engineer
  - On an agile team
  - Need to test it each build/iteration/sprint/etc.
- Automation Engineer / SDET / etc.

16

Automation: Considerations

- You need know system well enough to make meaningful automation
- If system is new, overheads are too high
  - So almost no automation in traditional environments
- If the same system is tested again and again
  - It makes sense to invest in setting up automation
- Automated interfaces should be stable enough
  - APIs are usually more stable on early stages

17

Time / Resource Considerations

- Performance tests take time and resources
  - The larger tests, the more
- May be not an option on each check-in
- Need of a tiered solution
  - Some performance measurements each build
  - Daily mid-size performance tests
  - Periodic large-scale / uptime tests outside CI

18

Automation: Limitations

- Works great to find regressions and check against requirements
- Doesn't cover:
  - Exploratory tests
  - Large scale / scope / duration / volume
- "Full Automation" doesn't look like a real option, should be a combination

19

Agenda

- Context-Driven Testing
- Early Performance Testing
  - Exploratory, Continuous
- *Environment / Scope / Granularity*
- Load generation
- Performance Testing / Engineering Strategy

20

Environment

- Cloud
  - No more excuse of not having hardware
- Lab vs. Service (SaaS) vs. Cloud (IaaS)
  - For both the system and load generators
- Test vs. Production

21

Scenarios

- System validation for high load
  - Outside load (service or cloud), production system
  - Wider scope, lower repeatability
- Performance optimization / troubleshooting
  - Isolated lab environment
  - Limited scope, high repeatability
- Testing in Cloud
  - Lowering costs (in case of periodic tests)
  - Limited scope, low repeatability

22

Find Your Way

- If performance risk is high it may be a combination of environments, e.g.
  - Outside tests against the production environment to test for max load
  - Lab for performance optimization / troubleshooting
  - Limited performance environments to be used as part of continuous integration

23

Scope / Granularity

- System level
- Component level
  - Service Virtualization, etc.
- Server time
- Server + Network (WAN simulation, etc.)
- End-to-end (User Experience)
- Each may require different approach / tools

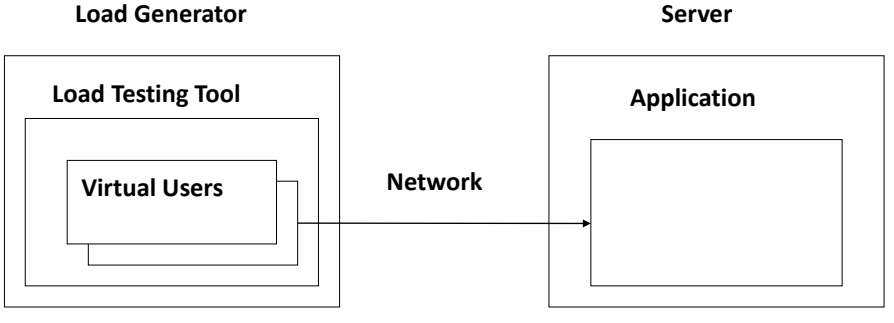
24

Agenda

- Context-Driven Testing
- Early Performance Testing
  - Exploratory, Continuous
- Environment / Scope / Granularity
- *Load generation*
- Performance Testing / Engineering Strategy

25

Record and Playback: Protocol Level



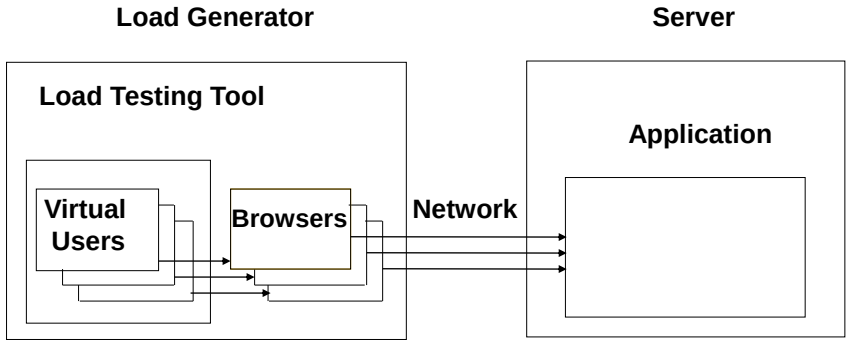
26

Considerations

- Usually doesn't work for testing components
- Each tool support a limited number of technologies (protocols)
- Some technologies are very time-consuming
- Workload validity in case of sophisticated logic on the client side is not guaranteed

27

Record and Playback: UI Level



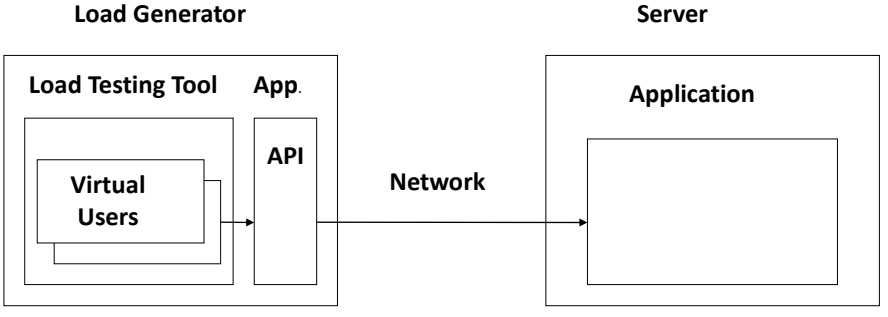
28

Considerations

- Scalability
  - Still require more resources
- Supported technologies
- Timing accuracy
- Playback accuracy
  - For example, for HtmlUnit

29

Programming



30

Considerations

- Requires programming / access to APIs
- Tool support
  - Extensibility
  - Language support
- May require more resources
- Environment may need to be set

31

Production Workload

- A/B testing, canary testing
- Should work well if
  - homogenous workloads and a way to control them precisely
  - potential issues have minimal impact on user satisfaction and company image and you can easily rollback the changes
  - fully parallel and scalable architecture

32

Agenda

- Context-Driven Testing
- Early Performance Testing
  - Exploratory, Continuous
- Environment / Scope / Granularity
- Load generation
- *Performance Testing / Engineering Strategy*

33

Performance Risk Mitigation

- Single-user performance engineering
  - Profiling, WPO, single-user performance
- Software Performance Engineering
  - Modeling, Performance Patterns
- Instrumentation / APM / Monitoring
  - Production system insights
- Capacity Planning/Management
  - Resources Allocation
- Continuous Integration / Deployment
  - Ability to deploy and remove changes quickly

34

Defining Performance Testing Strategy

- What are performance risks we want to mitigate?
- What part of this risks should be mitigated by performance testing?
- Which performance tests will mitigate the risk?
- When we should run them?
- What process/environment/approach/tools we need in our context to implement them?

35

Examples

- Handling full/extra load
  - System level, production[-like env], realistic load
- Catching regressions
  - Continuous testing, limited scale/env
- Early detection of performance problems
  - Exploratory tests, targeted workload
- Performance optimization/investigation
  - Dedicated env, targeted workload

36

Summary

- Testing strategy became very non-trivial
  - A lot of options along many dimensions
  - Defined by context
- "Automation" is only one part of it
  - Important for iterative development
- Part of performance engineering strategy
  - Should be considered amongst other activities

37

Questions?

Alexander Podelko

alex.podelko@oracle.com
alexanderpodelko.com/blog
@apodelko