Java Performance Concepts

Sang Shin
Michèle Garoche
www.javapassion.com
“Learn with Passion!”

Agenda

- Performance metrics
- Monitoring and profiling
- Development process

Before Getting Started

- Performance tuning is largely an art.
- There is no one approach that is always necessarily the right approach.
- There are performance issues which will require very specialized expertise to identify the root cause and be able to recommend a solution.
- Keep in mind that not every change made to improve performance will work for every customer environment.
- Performance testing must be integrated into the entire product lifecycle, development, build, release and sustaining to ensure continuous product quality.

Performance Metrics

Performance Testing

- Performance testing is defined as the technical investigation done to determine or validate the speed, scalability, and/or stability characteristics of the product under test.
- Performance-related activities, such as testing and tuning, are concerned with achieving response times, throughput, and resource-utilization levels that meet the performance objectives for the application under test.

Key Types of Performance Testing

- Performance test
  - To determine or validate speed, scalability, and stability.
- Load test
  - To verify application behavior under normal and peak load conditions.
- Stress test
  - To determine or validate an application's behavior when it is pushed beyond normal or peak load conditions.
- Capacity test
  - To determine how many users and/or transactions a given system will support and still meet performance goals.

Performance Metrics

- Throughput
- Runtime response time
- Memory Footprint
- CPU Resources
- Perceived performance (for GUI/CLI applications)

Perceived Performance

- End-user experience on GUI applications:
  - Rarely measure performance with a stopwatch
  - How fast something *feels*, not how fast it is
  - Response time of 100ms or less is perceived by the user as fast
  - Response time of 10 seconds or more is perceived as nonresponsive or hung
- Ways to improve how fast your users feel without actually making anything run faster
  - Changing the mouse cursor to a waiting cursor
  - Using multiple background threads and displaying a status queue for long running tasks
  - Showing a progress bar for relatively short tasks
  - When enumerating large sets of data always provide a filtering mechanism

Perceived Performance

- Start up time:
  - Lazy initialization is often useful.
  - Applets:
    - Use Jar files to minimize requests.
    - Install on client system if possible.
    - Obfuscators and size reduction tools.
    - Run empty applet to get VM loaded.
  - Applications:
    - Separate initialization thread.
    - Minimize dependencies for start screen.

Perceived Performance

- CLI applications
  - Run time is usually dominated by JVM initialization time for simple requests.
    - Provide an option to bundle multiple transactions into one request via text file or XML.
    - Provide a shell-like interface to the CLI so multiple commands can be issued in one session.
  - When displaying large lists of data, display the data in batches instead of waiting for the entire data set to be retrieved.

Monitoring and Profiling

Definition: Performance Monitoring

- An act of *non-intrusively* collecting or observing performance data from an operating or running application.
- Typically a “preventative” or “proactive” type of action. But, could be an initial step in a reactive action.
- Can be performed in production, qualification or development environments.
- Helps identify or isolate potential issues without having a severe impact on runtime responsiveness or throughput.
- Often times *monitoring* crosses over into trouble-shooting or service-ability.

Definition: Performance Profiling

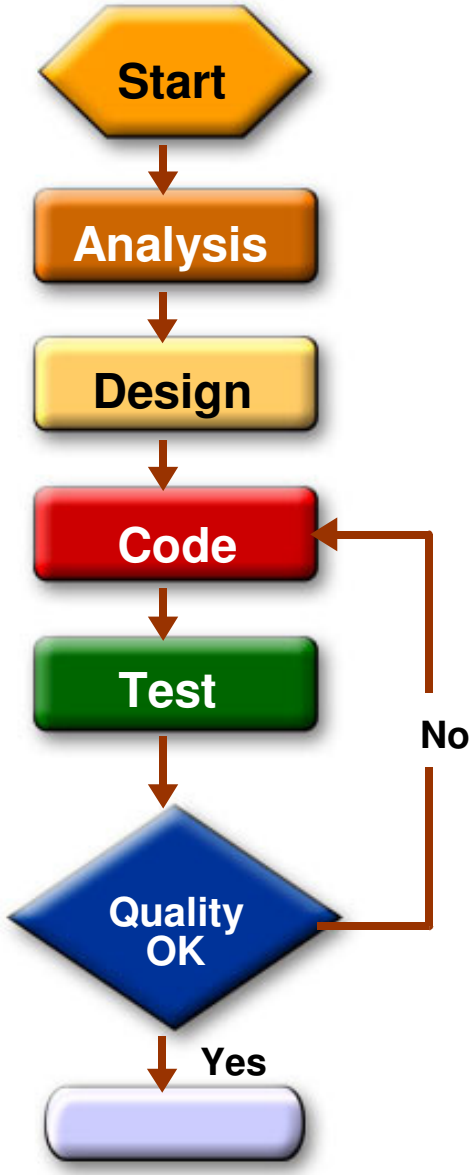
- An act of collecting or observing performance data from an operating or running application.
- Usually more intrusive than monitoring.
- Usually a narrower focus than monitoring.
- Typically a reactive type of activity. Could be a proactive activity in situations where performance is a well defined systemic quality or requirement for a target application.
- Seldom performed in production environments.
- Commonly done in qualification, testing or development environments.

Definition: Performance Tuning

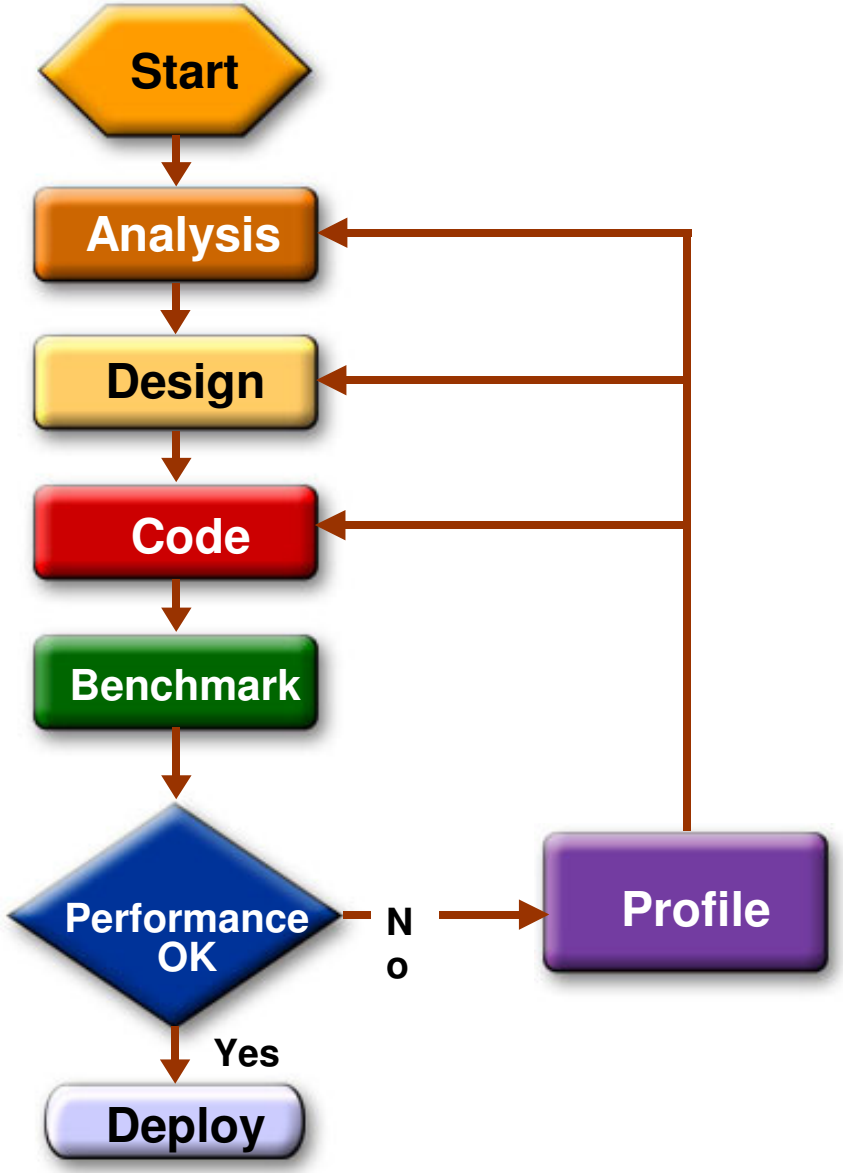
- An act of changing tune-ables, source code and/or configuration attribute(s) for the purposes of improving application responsiveness and/or application throughput.
- Usually results from monitoring and/or profiling activities.

Development Process

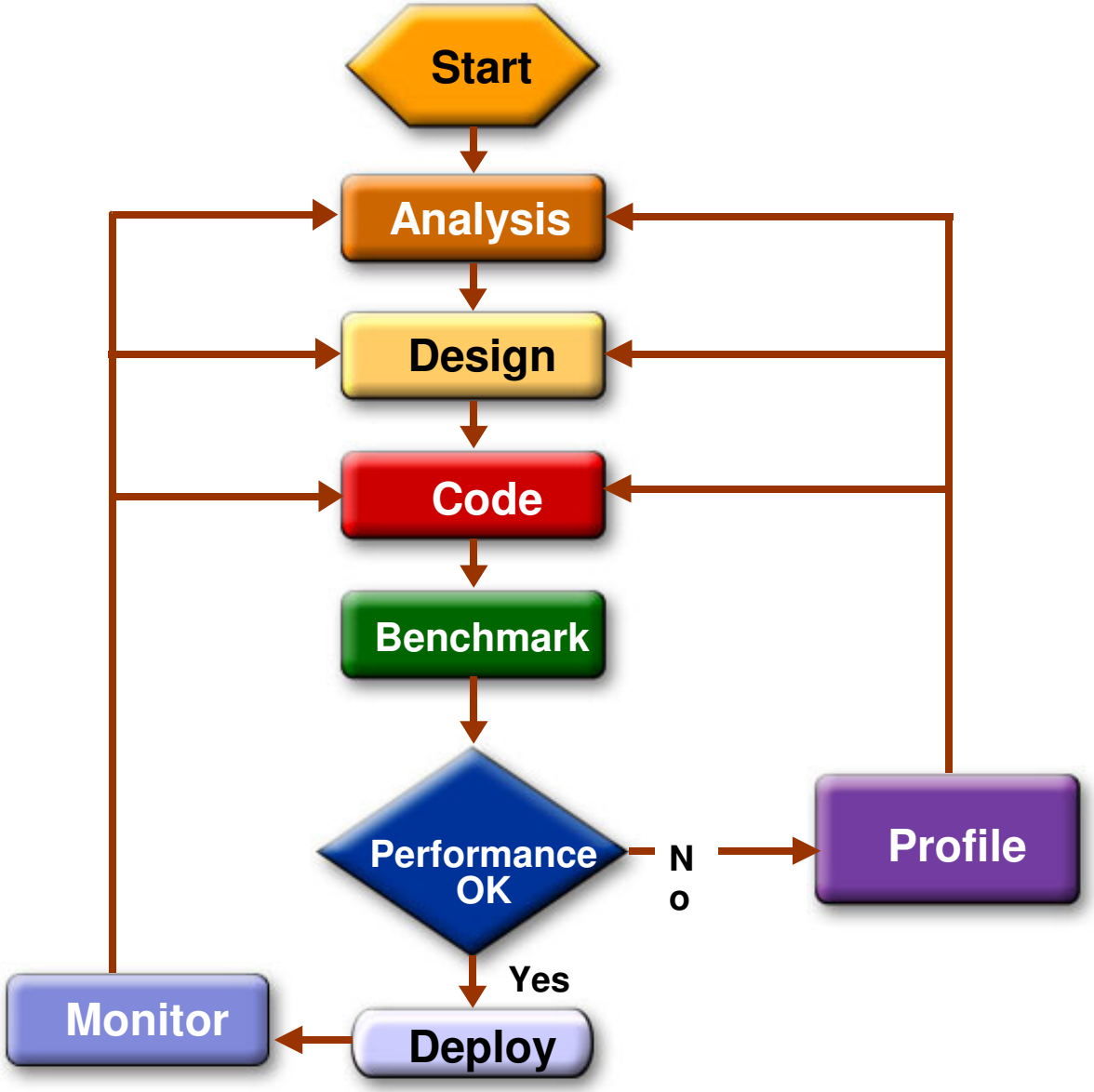
Typical Development Process



Application Performance Process



Application Performance Process



Performance Testing Activities

1. Identify the test environment
2. Identify performance acceptance criteria
3. Plan and design tests
4. Configure the test environment
5. Implement the test design
6. Execute the test
7. Analyze results, report, and retest

1. Identify the test environment

- Identify the physical test environment and the production environment as well as the tools and resources available to the test team.
- The physical environment includes hardware, software, and network configurations.
- Having a thorough understanding of the entire test environment at the outset enables more efficient test design and planning and helps you identify testing challenges early in the project.
- In some situations, this process must be revisited periodically throughout the project's life cycle.

2. Identify Performance Acceptance Criteria

- Identify the response time, throughput, and resource utilization goals and constraints.
- In general, response time is a user concern, throughput is a business concern, and resource utilization is a system concern.
- Additionally, identify project success criteria that may not be captured by those goals and constraints; for example, using performance tests to evaluate what combination of configuration settings will result in the most desirable performance characteristics.

3. Plan and design tests

- Identify key scenarios, determine variability among representative users and how to simulate that variability, define test data, and establish metrics to be collected.
- Consolidate this information into one or more models of system usage to be implemented, executed, and analyzed.
- A subset of the most important scenarios should be identified and incorporated into the build process for continuous reporting throughout the product lifecycle.

4. Configure the test environment

- Prepare the test environment, tools, and resources necessary to execute each strategy as features and components become available for test.
- Ensure that the test environment is instrumented for resource monitoring as necessary.

5. Implement the test design

- Develop the performance tests in accordance with the test design.

6. Execute the Test

- Run and monitor your tests.
- Validate the tests, test data, and results collection.
- Results must be repeatable and consistent in order for the testing to be considered valid.
- Test environments such as VM or Cloud may present challenges due to unpredictability of resource allocation.
- Execute validated tests for analysis while monitoring the test and the test environment.

7. Analyze Results, Report, and Retest.

- Consolidate and share results data.
- Analyze the data both individually and as a cross-functional team.
- Reprioritize the remaining tests and re-execute them as needed.
- When all of the metric values are within accepted limits, none of the set thresholds have been violated, and all of the desired information has been collected, you have finished testing that particular scenario on that particular configuration.

Thank you!

Check JavaPassion.com Codecamps!
http://www.javapassion.com/codecamps
"Learn with Passion!"