ECSE Software Validation Term Project

Part C Non-Functional Testing of Rest API

Two types of non-functional testing will be implemented in part C of the project. Performance Testing using Dynamic Analysis and Static Analysis of the source code.

Performance Testing

Create a program which adds a specific number of to-dos, contexts, or projects. Populate these objects with random data.

Modify unit test scripts to log time T1 from beginning to end of test including assessing correctness, set up and restore activities, and time T2 which is the time of the transaction being tested but without set up, tear down and assessing correctness. Log the real time of each sample.

Perform the following experiments:

- As the number of to-dos increase measure the time to add, delete or change a to do.
- As the number of projects increase measure the time to add, delete or change a project.
- As the number of contexts increase measure the time to add, delete or change a context.

Use operating system tools or open source tools to track CPU percent use and Available Free Memory while each experiment takes place, again log this information with the real time of the sample.

Operating system tools that may support dynamic analysis include:

```
Windows perfmon

Mac Activity Monitor

Linux vmstat
```

Static Analysis

The source code of the Rest API to do list manager will be studied with the community edition of the Sonar Cube static analysis tool.

```
https://www.sonarqube.org/downloads/
```

Recommendations should be made regarding changes to the source code to minimize risk during future modifications, enhancements, or bug fixes. Teams should look at code complexity, code statement counts, and any technical risks, technical debt or code smells highlighted by the static analysis.

Performance Test Suite Video

Show video of all performance tests running in the teams selected development environment.

Written Report

Target report size is between 5 and 10 pages.

Summarizes deliverables.

Describes implementation of performance test suite.

Include excel charts showing transaction time, memory use and cpu us versus sample time for each experiment.

Include excel charts showing transaction time, memory use and cpu us versus number of objects for each experiment.

Summarize any recommendations for code enhancements related to results of performance testing.

Highlight any observed performance risks.

Describes implementation of static analysis with Sonar Cube community edition.

Include recommendations for improving code based on the result of the static analysis.

Summary of clean code guidelines from Bob Martin.

Understandability tips

- Be consistent. If you do something a certain way, do all similar things in the same way.
- Use explanatory variables.
- Encapsulate boundary conditions. Boundary conditions are hard to keep track of. Put the processing for them in one place.
- Prefer dedicated value objects to primitive type.
- Avoid logical dependency. Don't write methods which works correctly depending on something else in the same class.
- Avoid negative conditionals.

Names rules

- Choose descriptive and unambiguous names.
- Make meaningful distinction.
- Use pronounceable names.
- Use searchable names.
- Replace magic numbers with named constants.
- Avoid encodings. Don't append prefixes or type information.

Functions rules

- Small.
- Do one thing.
- Use descriptive names.
- Prefer fewer arguments.
- Have no side effects.
- Don't use flag arguments. Split method into several independent methods that can be called from the client without the flag.

Comments rules

- Always try to explain yourself in code.
- Don't be redundant.
- Don't add obvious noise.
- Don't use closing brace comments.
- Don't comment out code. Just remove.
- Use as explanation of intent.
- Use as clarification of code.
- Use as warning of consequences.

Source code structure

- Separate concepts vertically.
- Related code should appear vertically dense.
- Declare variables close to their usage.
- Dependent functions should be close.
- Similar functions should be close.
- Place functions in the downward direction.
- Keep lines short.
- Don't use horizontal alignment.
- Use white space to associate related things and disassociate weakly related.
- Don't break indentation.
- Objects and data structures
- Hide internal structure.
- Prefer data structures.
- Avoid hybrids structures (half object and half data).
- Should be small.
- Do one thing.
- Small number of instance variables.
- Base class should know nothing about their derivatives.
- Better to have many functions than to pass some code into a function to select a behavior.
- Prefer non-static methods to static methods.

Tests

- One assert per test.
- Readable.
- Fast.
- Independent.
- Repeatable.

Avoid Code smells

- Rigidity. The software is difficult to change. A small change causes a cascade of subsequent changes.
- Fragility. The software breaks in many places due to a single change.
- Immobility. You cannot reuse parts of the code in other projects because of involved risks and high effort.
- Needless Complexity.
- Needless Repetition.
- Opacity. The code is hard to understand.