Test Impact Analysis

In this section:

Overview

The test impact analysis (TIA) functionality analyzes the coverage data for the application under test and generates a list of tests that have been affected by changes since the previous build. The list of tests are saved to a .lst file that you can pass as a resource to the SOAtest command line interface. SOAtest will only execute the subset of tests affected identified by TIA in order to validate the changes. The following overview describes the test impact analysis process:

1. **Configure the coverage agent** (agent.jar) shipped with SOAtest and attach it to your application under test (AUT).
2. **Configure your Test Configuration** to enable SOAtest to communicate with the agent and configure settings for creating the baseline coverage report.
3. **Run your full test suite** with the new configuration to collect information about what test cases cover and generate the baseline coverage report.
4. **Generate the .lst file containing the tests affected by changes**. When a new version of the application is available, run the `TestImpactAnalysis` script to process the baseline coverage report. A .lst file containing the tests affected by change will be created.
5. **Run the tests affected by change**. Deploy the latest version of the application (.war) to your server and run a job that executes SOAtest using the .lst file as the input to verify the changes.

Java Support

TIA is supported for Java 11 and higher.

Configuration

TIA is intended to be implemented as part of an automated process. Perform the following steps to enable TIA.

Deploy your Application

Deploy your application under test. An application packaged in any zip-based file format, including zip, war, jar, and ear, is supported, as is a hierarchical directory structure that contains standalone class files or class files embedded within a zip-based file.

Attaching the Coverage Agent to the AUT

SOAtest includes a Java agent that generates the coverage information necessary for SOAtest to determine which tests are affected by changes.

The agent is shipped in the <INSTALL>/test_impact_analysis/integration/coverage directory. It takes configuration settings from the agent.properties file in the same directory. You should copy the coverage directory that contains the agent.jar and agent.properties files to the machine where the AUT is running.

Configuring the Coverage Agent

Application servers usually contain more than one application. Additionally, common server classes or application libraries do not need to be instrumented. The agent only needs to collect coverage for application source code. Instrumenting all classes would be too time-consuming. For this reason, properly setting the scope of the coverage agent is very important.

You can configure the coverage agent by modifying the properties in the agent.properties and passing the properties to the `-javaagent` argument. The agent supports several parameters (see Coverage Agent Parameters), but configuring the default settings is suitable for most cases:

```
jtest.agent.serverEnabled=true
jtest.agent.includes=com/myapp/data,com/myapp/common/**
jtest.agent.excludes=com/myapp/transport/*,com/myapp/autogen/**
jtest.agent.associateTestsWithCoverage=true
jtest.agent.autostart=false
```

Coverage Agent Parameters

The following table describes all properties that can be set for the agent:

`jtest.agent.associateTestsWithCoverage`	Enables/disables associating coverage with particular tests; the default value is `false`.
`jtest.agent.runtimeData`	Specifies a location on the application server for the agent to store the coverage data it collects at runtime.
`jtest.agent.includes`	A comma-separated list of patterns that specify classes to be instrumented. The following wildcards are supported: `*` matches zero or more characters `**` matches multiple directory levels In the following example, all classes from the `com.myapp.data` package and all classes from package and subpackages that start with `com.myapp.common` will be instrumented: `com/myapp/data/*,com/myapp/common/**`
`jtest.agent.excludes`	A comma-separated list of patterns that specify classes to be excluded from instrumentation. The following wildcards are supported: `*` matches zero or more characters `**` matches multiple directory levels In the following example, all classes from the `com.myapp.transport` package and all classes from package and subpackages that start with `com.myapp.autogen` will be excluded from instrumentation: `com/myapp/transport/*,com/myapp/autogen/**`
`jtest.agent.autostart`	Enables/disables automatic runtime data collection; the default is `true`.
`jtest.agent.port`	Sets up agent communication port; the default is `8050`.
`jtest.agent.debug`	Enables/disables verbose output to console; the default is `false`.
`jtest.agent.collectTestCoverage`	Enables/disables collecting coverage information for test cases; the default is `false` .
`jtest.agent.enableMultiuserCoverage`	Enables/disables collecting web application coverage for multiple users; the default is `false`.
`jtest.agent.serverEnabled`	Activates the agent.
`jtest.agent.enableJacoco`	Enables the agent to collect coverage using the JaCoCo engine; the default is `false`.

When the properties are configured, add a `-javaagent` argument when starting your application server to attach the agent and include the agent configuration file:

```
-javaagent:'/path/to/agent.jar'=settings='/path/to/agent.properties',runtimeData='/path/to/runtime_coverage'
```

For your convenience, the coverage directory includes a script that will generate the `-javaagent` arguments. Run either the agent.sh or agent.bat script and copy the output to your server startup script.

```
$ ./agent.sh
Add this Java VM args to your process:
---------------------------------------------------
-javaagent:"/home/TIA/test_impact_analysis/integration/coverage/agent.jar"=settings="/home/TIA
/test_impact_analysis/integration/coverage/agent.properties",runtimeData="/home/TIA/test_impact_analysis
/integration/coverage/runtime_coverage"
---------------------------------------------------
Press any key to continue . . .
```

In the following example, the agent is attached to a Tomcat server with a JAVA_OPTS variable at the beginning of the catalina.sh (Linux) or catalina.bat (Windows) scripts:

Linux and macOS

```
if [ "$1" = "start" -o "$1" = "run" ]; then
JAVA_OPTS='-javaagent:"/home/TIA/test_impact_analysis/integration/coverage/agent.jar"=settings="/home/TIA
/test_impact_analysis/integration/coverage/agent.properties",runtimeData="/home/TIA/coverage_storage"'
fi
```

Windows

```
if "%1"=="stop" goto skip_instrumentation
set JAVA_OPTS=-javaagent:"C:\TIA\test_impact_analysis\integration\coverage\agent.jar"=settings="C:
\TIA\test_impact_analysis\integration\coverage\agent.properties",runtimeData="C:\TIA\coverage_storage"
:skip_instrumentation
```

Start the application and verify that the agent is ready by opening `<host>:8050/status` in your browser. You should see a JSON object that contains test, runtime_coverage, and testCase properties, e.g.:

```
{"test":null,"session":"runtime_coverage_20191008_1537_0","testCase":null}
```

You can also check the directory you specified with the runtimeData property (`/home/TIA/coverage_storage` in the example above). The directory should contain a set of static coverage data files. The files are generated when the agent is started.

Configuring the Test Configuration

Configure a Test Configuration to report test execution information to the coverage agent and generate the baseline coverage report. You can configure an existing Test Configuration or create and configure a new one (see Creating a Custom Test Configuration).

1. Open the Test Configuration and click the **Execution> Application Coverage** tab.
2. Enable the the **Collect application coverage** option and specify the host name or IP address where the application under test and coverage agent are hosted and the port number of the agent. The port number should match the value of the `jtest.agent.port` setting in the agent. properties file (default is `8050`). You can click **Test Connection** to verify that SOAtest can communicate with the coverage agent.
3. (Optional) Under **Coverage agent user ID**, you can specify a user ID so that coverage results can be associated with a specific user. A user ID should only be specified when application coverage for multiple users is enabled for the coverage agent.
4. Enable the **Generate baseline coverage report for test impact analysis** option and specify the following:
   - **Application binaries location** - The location that contains binaries of the application under test. You can specify the path to a folder or a .war, . jar, .zip, or .ear file.
   - **Report location** - The output directory of the baseline coverage report.
5. Enable the **Report coverage agent connection failures as test failures** option for test failures to be reported when the coverage agent connection fails, or when the application binaries location or report location are misconfigured. Otherwise connection problems will be reported to the console, but will not cause the test to fail.
6. Click **Apply** to save your changes.

Collecting Test and Coverage Data

Run your full test suite using the new Test Configuration to collect the data and generate the baseline coverage report.

If you already have an automated test run, you can modify the existing job to use the new Test Configuration, e.g.:

```
soatestcli.exe -data <your workspace> -resource <your tests> -localsettings <properties file with SOAtest
settings> -config <new test configuration>
```

See Testing from the Command Line Interface - soatestcli for details about building test execution commands with SOAtest. You can also manually run tests from the SOAtest GUI.

TEMP Directory for Linux/MacOS

Generating the baseline coverage report when the Test Configuration is run requires a temporary directory to be configured. If the TEMP variable is not already set, you may need to modify the TestImpactAnalysis.sh script located in the <INSTALL_DIR>/test_impact_analysis directory, for example:

```
#!/bin/bash
TMP=/tmp
```

Generating List of Tests Affected by Change

When a new version of the application is available, run the TestImpactAnalysis.bat (Windows) or TestImpactAnalysis.sh (Linux/macOS) script located in the <INSTALL_DIR>/test_impact_analysis directory using the following syntax:

```
TestImpactAnalysis.sh -app <path to new .war> -coverageReport <path to coverage.xml report> -outputLst <path to
.lst>
```

- The `-app` flag specifies the new deployable .war.
- The `-coverageReport` flag should specify the baseline *coverage.xml* report generated in the directory configured in your Test Configuration with the *Report location* option (see Configuring the Test Configuration).
- The `-outputLst` flag is optional and specifies where to output the results.

When the script finishes, the result will be saved to a .lst file in the directory specified with the `-outputLst` flag. If the flag is not included, the file will be saved to the <USER_HOME>/parasoft/test_impact_analysis/lsts/ directory by default. The default name of the .lst file will be `<yyyyMMdd_HHmmss>_affected_tests.lst`. The .lst file will only be generated if tests are affected by the changes.

Executing Tests Affected by Change

To execute only the tests reported by TIA, specify the .lst file during SOAtest execution with the `-resource` parameter:

```
soatestcli.exe -data <your workspace> -resource <path to .lst> -localsettings <properties file with SOAtest
settings> -config <your team's test configuration>
```

See Testing from the Command Line Interface - soatestcli for details about building test execution commands with SOAtest.