Checklist

Introduction of the Test Gap Analysis in .NET Projects

2017-01-18

Fabian Streitel
Dr. Andreas Göb
Dr. Elmar Jürgens
Dr. Dennis Pagano

Table of Contents

1 SUMMARY 3

2 PROJECT 3

3 OPERATIONS / DEVOPS 3

4 CQSE 4

5 PROFILER INSTALLATION 4

5.1 THE COVERAGE PROFILER 4

5.2 THE HEARTBEAT APPLICATION 4

5.3 INSTALLATION 5

5.3.1 Determining Which Profiler DLL to Use 5

5.3.2 Registering the Profiler for the Target Application 6

5.4 WEB APPLICATIONS 6

5.5 WHEN WILL THE REPORT FILE BE WRITTEN? 7

5.6 AZURE CLOUD 7

5.7 .NET CORE 7

5.8 TROUBLESHOOTING 8

6 BUILDS 8

1 Summary

Test Gap analysis is a tool, which supports the test service by bringing more transparency to the testing process. To this end, changes within an application are identified and matched against information about test execution. From a technical perspective, this requires two types of information: (1) the source code at the current revision and (2) execution information, which is collected via the Coverage profiler.

This document describes the necessary steps to introduce the Test Gap analysis in a .NET project. These steps are grouped by responsibilities. First, we detail necessary information that must be provided by the project. Then, technical measures are described that must be taken by Operations (or DevOps). Finally, the actual creation of the dashboards by CQSE is described. The last chapter is concerned with the installation of the coverage profiler.

2 Project

The project must provide the following information to CQSE:

- Definition whether the application has a client and/or server part and which of those should be included in the analysis. Depending on this, the profiler will need to be installed on the server and/or the client machines.
- Which .NET framework version is used. This determines the profiler version to be used.
- Whether the application is compiled as 32bit or 64bit.
- Description of the test environments. Here the most interesting information is, whether there are areas that should be analyzed separately, e.g. tests done by the developers, tests done by the test service or user acceptance tests. In this case, we usually provide one dashboard per area plus one overall dashboard.
- Description of the test process. This determines, whether the profiler must be registered in a start script or by setting user-specific environment variables. Furthermore, it must be determined if there are longer-running tests (e.g. overnight) that must be treated specially to get up-to-date results every day.
- Incrementing version numbers in the build. The analysis can only reliably distinguish between different versions of the application if each version of the application has its own distinct version number. These numbers must be in ascending order.
- Generating PDB files in the build. The analysis needs the PDB files to be able to map the coverage data from the binary files back to the source code. See the below section on builds for details.
- Select exclusion criteria. Some assemblies or types might be irrelevant for the Test Gap analysis and therefore must be excluded. Examples are test code, which does not need to be tested itself, or automatically generated types whose code is deliberately used only partially, that do not change over time and were generated by a trustworthy program.

3 Operations / DevOps

Operations / DevOps must perform the following tasks:

- Installation of the profiler on the test machines (client and/or server, see section 5)
- Facilitating access to the traces and binaries. The dashboard can read this data either from a network share of the project or the data can be deposited directly onto the TGA network share.

- For client applications: installation of the Heartbeat application and activation of the profiler. Setup of a time-based execution. This enables a regular, automatic check of whether a client machine is able to send trace data to the dashboard.

4 CQSE

Once all the above-mentioned information is available and the Operations tasks have been completed, CQSE will create the dashboards:

- Creation of one or more dashboards, depending on the test environment.
- Customization of the dashboard, in order to collect the necessary data from the network share and start the preprocessing of the data.
- For client applications: Creation of a Heartbeat dashboard and listing of all clients from which a regular Heartbeat trace is expected.
- Setup of the exclusion criteria as parameters of the dashboard.

5 Profiler Installation

5.1 The Coverage Profiler

The coverage profiler writes method coverage information of a .NET application into a report file (a.k.a. “trace file”). It uses the .NET profiling interface and therefore can be used with any .NET application. Depending on the .NET version in use (especially before 4.0), an older version of the profiler might be required.

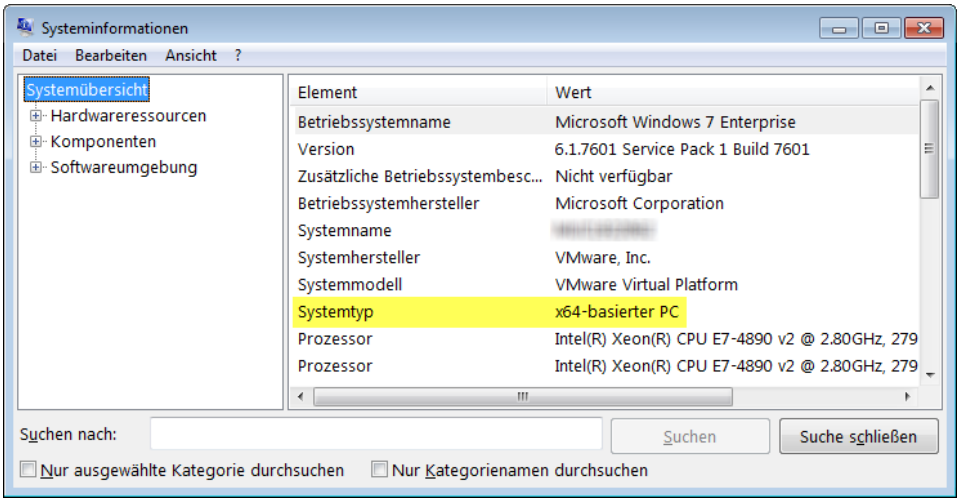
5.2 The Heartbeat Application

In addition to the profiler, a console application is provided that can be used for testing purposes, since it starts very fast and produces small trace files when profiled. Furthermore, this application can be used to regularly create small traces on client machines, e.g. for a daily test whether the profiler and the copy scripts are working.

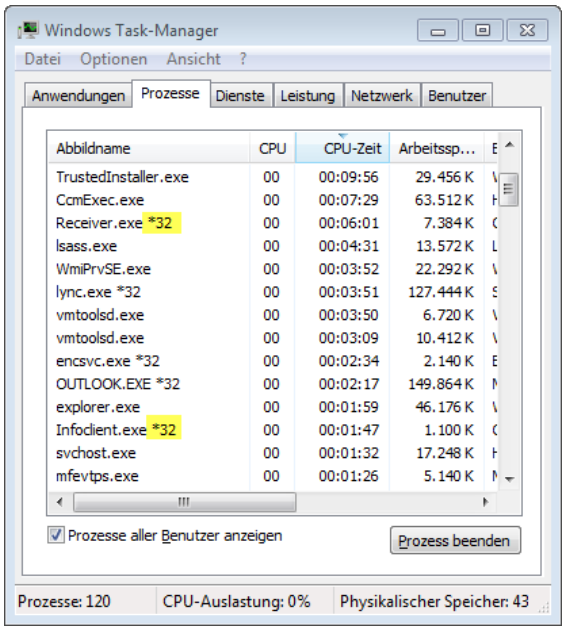
5.3 Installation

5.3.1 Determining Which Profiler DLL to Use

Before installing the profiler, it must be determined whether the x32 or x64 variant must be used. This does **not** always correspond to the installed operating system, but to the mode your application is executed in. Notably, if the application is started in 32bit mode on a 64bit machine, the 32bit version of the profiler must be used. In order to find out which profiler is the correct one for your application, you may use the “System Information” program that ships with Windows and look for “system type”.



- “x86-based PC” → use Profiler32.dll
- “x64-based PC” → check the Task Manager for your application’s process. If your application runs in IIS, look for the IIS user mode worker process *w3wp.exe*.



- Entry shows *32 → use Profiler32.dll
- Entry does *not* show *32 → use Profiler64.dll

5.3.2 Registering the Profiler for the Target Application

From the last section, you should know which profiler is the right one in your setting. For simplicity, the installation of the 32bit profiler is described in the following. If you need the 64bit profiler, the name of the DLL file must be adjusted accordingly. If the application is run in a web application server, please make sure to also read section 5.4.

The profiler can be installed into any directory on the target machine, but we recommend the following steps:

1. Create directory C:*Program Files**Coverage Profiler.*
2. Copy the *Profiler32.dll* file and the Heartbeat application to this directory.
3. Create and share a target directory where trace files can be written, e.g. *C:**Users**Public**Traces*. The user whose account runs the analyzed application must have write access to this directory.
4. For client applications: Create a job that profiles the heartbeat application daily, including copying the created trace files into the shared directory.
5. Set the following environment variables for the user that starts the application or integrate the variables into the application's start scripts (using *set*):

Environment variable	Value	Description
COR_ENABLE_PROFILING	1 (or 0)	Enables or disables the profiler
COR_PROFILER	{DD0A1BB6-11CE-11DD-8EE8-3F9E55D89593}	The GUID of the DLL file
COR_PROFILER_PATH	C:\Program Files\Coverage Profiler\Profiler32.dll	Path to the DLL file
COR_PROFILER_TARGETDIR	C:\Users\Public\Traces	Target directory for the trace files
COR_PROFILER_LIGHT_MODE	1 (or 0), default is 0	Enable ultra-light mode
COR_PROFILER_EAGER_MODE	1 (or 0), default is 0	Enable eager mode (i.e. write to disk immediately). This should only be used in conjunction with light mode.

The light mode is only available in Profiler versions 0.9.2.5 or newer, which is itself only available for .NET 4.0 or newer. In addition, .NET before 4.0 ignores the variable COR_PROFILER_PATH, but needs the profiler DLL to be registered with the OS: *regsvr32.exe Profiler32.dll*

5.4 Web Applications

If the application is run in an IIS application server, the following steps need to be taken:

- Make the relevant application pool see the environment variables
- Regularly recycle the relevant application pool

The environment variables must be set either system-wide or preferably for the account that runs the respective IIS app pool. For the latter, the app pool needs to have "Load User Profile" enabled in the Advanced Settings, and the environment variables need to be set for that user as follows:

- Go to IIS Manager, Application Pools, [App Pool], Advanced Settings, Process Model – Load User Profile, set to True. Also see http://www.iis.net/learn/manage/configuring-security/application-pool-identities
- Set the environment in the Registry (http://dotnetmonitor.vlasovstudio.com/2012/01/19/monitoring-a-net-application-running-in-iis/)

Note that if the Application Pool is run under the *ApplicationPoolIdentity*, a special user will be created for just that single application pool. This only happens when Load User Profile is set to True after the pool is recycled once. These special pool users have SIDs that start with S-1-5-82-. Their profile image path ends with the name of the application pool.

As an alternative, the environment variables may be attached to the windows service running IIS instead of the user. Note that this causes ALL application pools to be profiled. This can be done in the registry by accessing
`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\[ServiceName]\Environment`
and setting the variables there. For IIS, the interesting services are W3SVC and IISADMIN.

IIS must be restarted after setting the environment variables in order for them to have any effect (e.g. by running *iisreset* in an administrator command prompt). On more recent IIS versions (tested with IIS 7), if you set the variables for the app pool account and not the service, recycling the application pool is enough and a full restart is not required.

After restarting the IIS/recycling the pool and opening the application in a browser, a trace file should be created. As web applications are long running processes, it must be taken into account that the report file is written only after the process has ended, e.g. after stopping IIS, after stopping the profiled web site, or after recycling the application pool (cf. Section “When will the Report File be Written”). In these cases, it must be ensured that the application is stopped and restarted in regular intervals. In the case of IIS, we suggest to adjust the configuration to automatically recycle the application pool every night.

5.5 When will the Report File be Written?

The report file is created immediately after the first call of a .NET method. However, it remains mostly empty as long as the process runs. This avoids an unnecessary performance overhead due to file accesses. As soon as the process ends gracefully, the report is finished and all collected information is written to the report file.
In the case of long running processes it must be taken into account that the report file is written only after the process has ended. In such a case, it must be ensured that the application is stopped and restarted in regular intervals.

5.6 Azure Cloud

Azure instantly kills the application when it is stopped/restarted, thus the profiler has no time to write anything to the trace file. To remedy this, enable eager mode. Please note that eager mode should only be used together with light mode as it will otherwise drastically increase lag during the application startup.
To deploy the profiler to the Azure cloud, enable the FTP account. Store the profiler in a directory that will not be overwritten by a new deployment, e.g. D:\home\site\repository\profiler\
Also create a new directory to write the traces to, e.g. D:\home\site\repository\traces\
To set the environment variables in Azure, go to the application settings and add them as new application settings entries.

5.7 .NET Core

.NET core is fully supported but requires different environment variables to be set:

Environment variable	Value	Description
CORCLR_ENABLE_PROFILING	1 (or 0)	Enables or disables the profiler
CORCLR_PROFILER	{DD0A1BB6-11CE-11DD-8EE8-3F9E55D89593}	The GUID of the DLL file
CORCLR_PROFILER_PATH_32	C:\Program Files\Coverage Profiler\Profiler32.dll	Path to the 32bit DLL file
CORCLR_PROFILER_PATH_64	C:\Program Files\Coverage Profiler\Profiler64.dll	Path to the 64bit DLL file
COR_PROFILER_TARGETDIR	C:\Users\Public\Traces	Target directory for the trace files

The other variables are the same.

5.8 Troubleshooting

You must ensure that the profiled application has read permissions to the location of the profiler DLL and write permissions in the target directory (*COR_PROFILER_TARGETDIR*). If the target directory is not set, does not exist, or is not writable by the process, no trace file can be created. Profiling can also be tested by starting any .NET application from the console. However, in this case a new shell must be started for the new environment variables to take effect.

You can check whether the environment variables are visible to a certain process using the free Microsoft Process Explorer:

1. Open Process Explorer, and look for the respective process (e.g. the IIS user mode worker process w3wp.exe).
2. Right click on the process, click Properties and then click the Environment tab.
3. Look for the COR_ENABLE_PROFILING and COR_PROFILER values.

The .NET runtime will create an error message to the event log if loading the profiler fails. This means that the environment variables were noticed by the runtime but the profiler crashed before it could be used.

Things to check if no trace files are written:

- Are the environment variables set correctly (see the table above)?
- Are the environment variables set for the right user?
- Does the application process have read rights on the profiler DLLs and read/write rights on the trace folder?
- Did you choose the correct DLL (Profiler32.dll vs. Profiler64.dll)?
- IIS: Is the application pool set to pick up the environment of its user?
- IIS: Did you recycle the application pool?

6 Builds

In order to be able to interpret the generated coverage data, Teamscale needs access to the corresponding PDB files. These contain the mapping from the compiled assemblies back to the source code.
These files must be generated during your build process and must be available to Teamscale for every assembly in every version that is deployed to a test system.
The following steps need to be taken:

- Enable PDB file generation for your build by setting /p:DebugType=pdb in the MSBuild Arguments. It is recommended to set this once for the entire build and not on .csproj file level.
- Ensure that the PDB files are not deleted during the build process. If necessary, copy them to a separate location before packaging of your releases
- Make the location of the PDB files accessible to Teamscale

A specialized nightly job must be set up to pick up the PDB files and make them available to Teamscale.