The FaultCheck distribution with a QuickCheck example

Scripts

This distribution contains several scripts to automate certain tasks. Their purpose is as follows:

Directory: /

set_env
Set up the environment for running FaultCheck and the examples where the dynamic libraries are looked up and the includes are resolved.

build
Compiles FaultCheck and the C files for the example(s).

build_doc
Generates the documentation from the README.md markup file. The output appears in the *Documentation* directory.

clean
Removes the object files from the build process.

Directory: /Examples/AUTOSAR_E2E_QuickCheck

gen_coverage
Generate C code coverage reports in HTML and in LaTeX if gentex is installed.

clean_coverage
Remove the coverage files.

Installing dependencies

In order to run this software, make sure that all libraries that are required are installed. For FaultCheck itself, the Qt SDK is required and for the examples the GNU C compiler is required. For the FaultCheck tests, the sbt tool is required. To generate the coverage reports and the documentation, some additional tools are required. For the QuickCheck example, an erlang installation with QuickCheck is required.

Ubuntu Linux

Setting everything up on a fresh Ubuntu install is fairly simple. First, install some dependencies:

```
sudo apt-get install build-essential qt-sdk
```

QuickCheck Example
For the QuickCheck example to run, install Erlang and QuickCheck as usual.

Coverage information
To generate coverage information with the provided script for the examples, install lcov 1.10 or later. On Ubuntu 14.04, the version from the repositories can be used:

```
sudo apt-get install lcov
```

For earlier versions of Ubuntu, download and install an updated version of lcov from here:
http://packages.ubuntu.com/trusty/all/lcov/download

If you would like to have LaTeX output from lcov, download gentex from here:
https://github.com/jarikomppa/lcov-latex

ScalaCheck tests
For the ScalaCheck tests, install scala, openjdk and sbt:

```
wget http://apt.typesafe.com/repo-deb-build-0002.deb
sudo dpkg -i repo-deb-build-0002.deb
sudo apt-get update
sudo apt-get install sbt scala openjdk-7-jdk
```

Documentation
To generate LaTeX, PDF and HTML documentation from the markdown file, install pandoc:

```
sudo apt-get install pandoc texlive-latex-base cm-super
```

Setting up the environment

From the root directory, set up the environment with the provided script (note that source has to be used to run the script in the current shell):

```
source set_env
```

Build everything:

```
./build
```

At this point, the QuickCheck example and the functionality tests should work from the current shell. Note that the environment has to be set with the source command every time a new shell is started.

Running the QuickCheck example

If the AUTOSAR example is downloaded and present in the examples directory, go to *Examples/AUTOSAR_E2E_QuickCheck* and start erlang:

```
erl
```

Compile and run the example from the erlang shell:

```
c(airbag_eqc).
airbag_eqc:test(1000).
```

You should see lots of dots and explosion commands printed. In order to switch the E2E-library on and off, change the following define in *airbag_eqc.erl*:

```
-define(USE_E2E, yes).
```

When the E2E-library is not used, a large enough number of tests should reveal a failing test case or two.

Generation code coverage information

The *gen_coverage* script in the example can be used to generate coverage information in the *Coverage* directory. This can only be done after the experiment has been run.

Testing FaultCheck

There are some property-based tests that can be used to verify that FaultCheck works as intended. These tests are implemented in ScalaCheck. When new features are added to FaultCheck, these tests should be updated and/or re-run. Currently, there are two different tests: one for the probing-part of FaultCheck and one for the packet-based communication channel of FaultCheck. For these tests to work, everything has to be built and the environment has to be set up as described above.

Testing the probing part

Go to *Tests/General* and run the test with

```
sbt run
```

All the tests should pass.

Testing the communication channel

Go to *Tests/Packet* and run the test with

```
sbt run
```

All the tests should pass.

Generating documentation

Currently, this tutorial is written with the markdown syntax in the *README.md* file and the files in the Documentation directory are generated with the *build_doc* script. To update the documentation, edit the *README.md* file and run the script:

```
./build_doc
```

Make sure that the dependencies for this script are installed as described above.