Beagle

Quality Control Report

Annika Berger, Joshua Gleitze, Roman Langrehr,
Christoph Michelbach, Ansgar Spiegler, Michael Vogt

6th of March 2016

at the Department of Informatics
Institute for Program Structures and Data Organization (IPD)

Reviewer:	Jun.-Prof. Dr.-Ing. Anne Koziolek
Advisor:	M.Sc. Axel Busch
Second advisor:	M.Sc. Michael Langhammer

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

1 Tests

1.1 JUnit Tests

As of this writing (2016-03-06) we use about 400 automated JUnit tests to test Beagle on a per-method level. Every merge into the upstream master requires these tests to pass. For more details see 1.2.1.

1.2 Integration Tests

1.2.1 T200

Every time something is added to Beagle, Travis sets up an Ubuntu system with only the specified software installed. The combination of code already in the upstream master and the changes in question then has to pass all automated tests in order to be mergeable. Thus it is always guaranteed that Beagle as of its state in the upstream master passes all automated tests on a system with only the specified software installed.

1.2.2 T210

Beagle has been tested manually to work with Kieker.

1.2.3 T220

Beagle does obey the PCM standards when writing data and is not required for Palladio to access the PCM after it has been created.

1.2.4 OT200

Beagle has been tested on Linux and Windows operating systems.

1.3 Tests Defined in SRS

1.3.1 Mandatory

All mandatory tests defined in the SRS (/T10/ – /T60/) are tested either automatically using JUnit or manually. Instructions of how to proceed in manual tests and what to expect for a successful test can be found in `Manual ManualTests.MD`.

1.3.2 Optional

- /OT10/ and /OT20/ are tested automatically using JUnit tests.
- /OT50/ is partly tested with manual tests but a full test requires a bigger project which as of yet we don't have available.
- /OT40/ has not been tested because of practical reasons.
- The optional feature for /OT70/ has been implemented and was tested manually. Instructions for manual testing can be found in `Manual Tests.md`.
- /OT60/ has been deemed impractical because no practical way of combining old results with new results could be found.
- /OT100/ is inherently fulfilled in our approach.
- All optional tests not mentioned above are omitted because the features they are testing are optional features which have not been implemented.

2 Bugs

2.1 Implementation Phase

There were no known bugs or other undesired behaviour at the end of the implementation phase. We were not able to find any such during the quality assurance phase either. Therefore no bugs which originated in the implementation phase were fixed.

2.2 Quality Assurance Phase

Due to our policy to only merge branches into the upstream master if they are free of knows bugs and other undesired behaviour, we never introduced code known to contain errors into our upstream master. As of this writing (2016-03-06) there have been no occurrences of subsequent detection of bugs in upstream master either.

3 Changes

3.1 GUI

During the implementation phase a GUI consisting of a `jWizard` and a `jDialog` was created. During the quality assurance phase we removed the `jWizard` part of the GUI and replaced it by a launch configuration. The `jDialog` code remained the same but was moved from `GuiController.java` to `ProgressDialogController.java`. The launch configuration offers everything the `jWizard` solution offered and also makes it possible to deselect elements to analyse.

4 Additional Features

4.1 Adaptive Timeout

Implementing an adaptive timeout is an optional feature listed in the SRS. We consider an adaptive timeout a valuable extension of Beagle's original features because it allows evaluation to a reasonable degree of accuracy devoid of excessive CPU time consumption without the user being required to be able to estimate the required time (confer constant timeout).

An adaptive timeout based on linear regression was implemented first but later having the adaptive timeout based on the ageing algorithm was consistently agreed upon to make better predictions and to be better fitted for practice. Thus, Beagle now features an adaptive timeout based on the ageing algorithm.

4.2 Pausing and Continuing

We decided to implement the optional pausing and continuing feature. The user can pause / resume / abort the analysis via the buttons provided in the dialog shown while Beagle is running.

Pausing happens softly which means that the current `AnalysisController` loop round (as defined in the Software Requirements Specification) is finished and the `FinalJudge` is called before the thread goes to sleep. Contrary to this, aborting still lets the current measurement tool / `ProposedExpressionAnalyser` finish to avoid data inconsistencies but does not call the `FinalJudge`.