Beagle

Software Requirements Specification

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

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

Reviewer:
Second reviewer:
Advisor:

–

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

Contents

Reference notation

This document uses a fixed notation for all of its contents, making them referenceable:

/P#/ purpose criterion

/B#/ purpose boundary

/A#/ application attribute

/G#/ target group

/E#/ software environment attribute

/D#/ data

/F#/ functional requirement

/Q#/ non functional requirement

/C#/ challenge or assumption

/T#/ test case

/M#/ model

A preceding “O” marks optional points. These relate to features that are desired and planned, but can not surely be implemented in the project’s scope. They also serve as an outlook for further development.

1 Purpose and Goals

When developing software, specifying its architecture in a sophisticated way is a crucial, yet challenging task. Decisions made at this point often highly influence software's properties, such as maintainability and performance. Palladio is a software enabling developers to analyse performance and other such metrics of component-based software at the definition phase, before actually writing any code. To achieve that, the software is meta-modelled by the Palladio Component Model (PCM).

If no code exists yet, models are created completely manually. But in many scenarios some to all source code has already been written. In such cases however, analysis with Palladio might still be wished. SoMoX is a tool for static code analysis of source code, to re-engineer the software's architecture into a PCM. Unfortunately, SoMoX' results are limited, as they do not contain resource demands which are essential for performance analysis.

The purpose of this project is to analyse software dynamically by conducting performance measurements on its source code, in order to determine the resource demands of its component's internal actions. Adding this information to the software's PCM will enable developers to use Palladio to analyse and optimise their existing software with minimal effort.

1.1 Must Have Criteria

/P10/ Beagle shall enable the user to analyse given source code for its internal actions' resource demands.

/P20/ Beagle shall automatically annotate its resource demand findings in a given instance of the software PCM.

1.2 Nice to Have Criteria

/OP10/ Beagle should determine the approximate probability in SEFF conditions, which are annotated in the PCM, for each case. If the probability depends on the input parameters, Beagle should determine this dependency.

/OP20/ The branch probability as a function of the input parameters should be annotated in the PCM.

/OP30/ Beagle should determine in SEFF loops, which are annotated in the PCM, too, how often the loop body is executed approximately.

/OP40/ The number of loop executions as a function of the input parameters should be annotated in the PCM.

1.3 Boundary

/B10/ Beagle does not perform actual measurements on source code, as this is done by other software and transferred through the Common Trace API (CTA).

/B20/ Beagle does not reconstruct a model of software architectures from their source code, as this is done by other tools like SoMoX.

/B30/ Beagle does not reconstruct the internal structure of components like their service effect specification (SEFF), as this is done by other tools like SoMoX.

/B40/ Beagle only analyses Java programs.

/B50/ Beagle does no performance analysis or prediction, as this is done with Palladio.

2 Application

2.1 Application Field

/A10/ Beagle can be used for re-engineering of existing software. When starting to use Palladio for an existing software Beagle is helpful to detect bottlenecks and problems using the PCM SoMoX created. This allows an more efficient re-engineering.

/A20/ Beagle may be used for prototyping. As it is possible to create a PCM without existing code with the help of Palladio Beagle can analyse the yet unimplemented prototype. The PCM with the additional information from Beagle completes the prototype as it contains information about the resource demands for each component.

/A30/ Beagle can be used for software development. Early implementations of components can be analysed with Beagle in order to predict their performance in interaction with the software system. This leads to earlier detection of arising problems which hence can be fixed in time.

/A40/ Beagle can be used to verify design and implementation. After designing the software with Palladio and implementing it, SoMoX and Beagle can be used to compare. Differences and problems in implementation can be detected and resolved more easily.

2.2 Target Group

/G10/ Software architects will use Beagle predominantly for /A10/ and /A40/.

/G20/ System deployers will use Beagle predominantly for /A40/.

/G30/ Component developers will use Beagle predominantly for /A20/ and /A30/.

3 Environment

/E10/ Beagle requires Java and Eclipse to run.

/E20/ Beagle requires a PCM instance modelling the software to be analysed. The model must contain all components and their SEFFs.

/E30/ Beagle requires the CTA to communicate with performance measurement software.

4 Data

In the following section, “the software” refers to the software a user wants to analyse with Beagle. The term refers not only to source code, but also its conceptional attributes, like its purpose, structure and architecture. The software is excpected to be built using component-based software architecture. This assumption is with little loss of generality, as any software

4.1 Input

/D10/ The software’s source code. It must be written in Java and be compilable by the JDK installed on the PC Beagle runs on.

/D20/ The software’s components’ boundaries. Beagle uses the model to determine component boundaries. The model’s quality and accuracy thus have a direct impact on Beagle’s results’ quality and accuracy.

/D30/ The software’s components’ SEFF, contained in the PCM instance (/D20/). Beagle uses the SEFF information to determine internal actions and measurement points. Their quality and accuracy thus have a direct impact on Beagle’s results’ quality and accuracy.

/OD10/ User provided information about the software’s parts to be analysed.

/OD20/ User provided information about measurement timeouts. May be provided prior to or during Beagle’s execution.

4.2 Output

5 Functional Requirements

Given a software's source code together with a valid PCM instance correctly describing the software's components, including their SEFF, Beagle must fulfil the following requirements:

5.1 Measurement

/F10/ Using the information provided in the PCM, Beagle determines the sections in the source code to be measured in order to find internal actions' resource demands.

/F20/ Beagle conducts one or multiple measurement softwares to measure the sections found in /F10/.

/F30/ Beagle uses existing measurement software to conduct performance tests on the source code.

/F40/ Beagle supports the CTA to communicate with measurement software

/F50/ Beagle does not modify the provided source code files.

/F60/ Measurement results are saved onto a persistent medium to avoid data loss.

/OF10/ Beagle approximately determines relations between components' interface parameters and their resource demands.

/OF20/ Beagle determines the probability for each case to be taken in encountered SEFF conditions.

/OF30/ Beagle determines the probability for each case to be taken in encountered SEFF conditions depending on the component's interface parameters.

/OF40/ Beagle determines the probable number of repeats in encountered SEFF loops.

/OF50/ Beagle determines the probable number of repeats in encountered SEFF loops depending on the component's interface parameters.

/OF60/ The user may choose whether Beagle will analyse the whole source code or only parts of it.

/OF70/ The user may choose to re-test the source code or parts of it, in order to either gain more precision or to reflect source code changes.

/OF80/ The user may launch and control a test of a component over a network.

/OF90/ The user may pause a measurement of a component.

/OF92/ Beagle reports its progress back to the user.

/OF94/ The user may specify the parameterisation of a component.

5.2 Result Annotation

/F100/ Beagles stores all its results in the software's PCM. (Hereafter to be called "Beagle's result PCM instance").

/F110/ Beagle's result PCM instance is a valid PCM.

/F120/ As far as technically possible, Beagle's results can be read from Beagle's result PCM instance by a Palladio installation without Beagle.

/F130/ Beagle's result PCM instance contains all contained components' internal actions' resource demands.

/F140/ Any information found in the PCM instance provided to Beagle will be found in Beagle's result PCM instance.

/OF100/ Beagle annotates resource demands in its result PCM instance dependent on the component's interface's parameters using the PCM's expression language for resource demands.

/OF110/ Beagle's result PCM instance contains all contained SEFF conditions' estimated branch probability.

/OF120/ Beagle's result PCM instance contains all contained SEFF conditions' estimated branch probability dependent on the containing component's interface's parameters.

/OF130/ Beagle's result PCM instance contains all contained SEFF loops' estimated branch probability.

/OF140/ Beagle's result PCM instance contains all contained SEFF loops' estimated branch probability dependent on the containing component's interface's parameters.

todo Parameterisation of results

todo Analyse source code for its architecture and performance in one turn (e.g. with SoMoX & Kieker).

6 Non Functional Requirements

6.1 Dependencies

/Q10/ In order to use Beagle, the user is not required to have any software installed but Java, Eclipse, Palladio and a measurement software supported by Beagle.

/Q20/ Beagle does not depend on any specific measurement software.

/Q30/ Beagle does not require its input artefacts to be generated by a specific software.

/OQ10/ Beagle can be used on every combination of operating system and hardware platform Eclipse and Palladio runs on.

6.2 User Interface and Experience

/Q100/ Beagle is implemented as an Eclipse plugin. As both Palladio and its extensions are Eclipse plugins, this ensures good usability for users.

/Q110/ Beagle can be controlled by context sensitive menus in Eclipse.

/OQ100/ Beagle is integrated in SoMoX, such that it is automatically executed after SoMoX has finished.

/OQ110/ Beagle can obtain its input artefacts from SoMoX, such that the user does not need to provide further information after SoMoX was started. If Beagle requires more information than SoMoX provides, the user can already submit it while configuring SoMoX.

7 Test Cases

8 Discussion

8.1 Assumptions

/C10/ The measured software was built using component-based software architecture. This assumption is derived from working with Palladio, which was built for analysing component-based software. Fortunately, it most of the time imposes little loss of generality, as any object oriented software can be described using terms of component-based software architecture (regarding each class as a component in the worst case). Such software will naturally not have the advantages that come with the component-based software approach, but might still be analysed for their performance.

/C20/ The measured software has a constant, deterministic runtime for a fixed configuration of input parameters, when ignoring influences of the hardware, operating system and error of measurement. This will be the case for most software. The fact the user tries to measure the software when using Beagle implies he expects it to behave in such a manner.

8.2 Challenges

/C100/ For accurate measurements, the CPU of the test server needs to be controlled. This involves disabling turbo boost, reading the temperature of each core and making sure the CPU is in a real world application thermal state, and possibly further measures.

/C110/ Scalability and transferability need to be ensured.

/C120/ Different hardware platforms vary in different dimensions (CPU frequency, number of CPU cores, size and distribution of CPU caches, speed of RAM, etc.), yet the results have to be representative.

/C130/ Only one component is tested at a time, yet components need to interact with each other. This means that other components need to be mocked.

9 Models

9.1 Scenario

Imagine, that a Java based online shop is running on a middle-class web server of a company named “EmmaSun”. During the first few years the software could deal with almost 99,9% of the requests and orders that are handled quite well without any delay. After an enormous expansion since the last year, the user numbers are currently growing for about 5 percent each week. Although the current servers are designed to fulfil a distinctly higher amount of user requests, the administration reported some few dropouts as well as increasing waiting times in single applications. Unfortunately, the software is based on an early design that has grown over years with missing documentation in many cases. The effort to re-write the complete software is an impossible act. The only solution is, to reanalyse the software’s source code and hopefully find any bottlenecks that can be repaired with lesser effort. But reanalysing source code is also a quite unmanageable task. So at this point, Beagle is used. Beagle helps the team of software architects that was commissioned by EmmaSun to analyse the whole software.

Terms and Definitions

Common Trace API an API developed by NovaTec GmbH for measuring the time, specific code sections need to be executed. . 2

component an artifact of a software development process with a description of its application.. 1, 2, 5, 7, 9–11, 17

component developer builds composite components, specifies components, interfaces, and data types, and specifies data types. Creates service effect specifications, stores modelling and implementation artecfacts in repositories, and implements, tests, and maintains components. . 3

component-based software a software constituted of components. . 1, 17

component-based software architecture a software architecture utilising the concept of component-based software, therefore taking advantage of the reusability of its parts and preserving the same for newly created components. . 7, 17

CTA Common Trace API. 2, 5, 9

internal action sequence of commands a component executes without leaving its scope (e.g. without calling other components). 1, 7, 9, 10

Kieker a Java-based application performance monitoring and dynamic software analysis framework.

[3] A measurement software Beagle aims to support. . 11

measurement software software capable of measuring the time, given source code needs to execute some task. The software's results are usually returned in a time unit like nanoseconds. Beagles interacts with such software through the CTA and uses it to find resource demands. . 9, 13

Palladio an approach for the definition of component-based software architectures with a special focus on performance properties. . 1–3, 13, 17

Palladio Component Model a domain-specific modelling language (DSL) used by Palladio.

It is designed to enable early performance predictions for software architectures and is aligned with a component-based software development process.

[2] . 1

PCM Palladio Component Model. 1–3, 5, 7, 9–11

resource demand how much of a certain resource—like CPU, Network or hard disk drive—a component needs to offer a certain functionality. A resource demand is ideally specified platform independently, e.g. by specifying required CPU cycles, megabytes to be read, etc. If such information is not available, resource demands can be expressed platform dependent, e.g. in nanoseconds. In this case, a certain degree of portability can still be achieved if information about the used platforms' speed relative to each other is available. . 1, 3, 9, 10

SEFF service effect specification. 2, 5, 7, 9

SEFF condition conditions (like Java's `if`, `if-else` and `switch-case` statements) which affect the calls a component makes to other components. Such conditions are—contrary to conditions that stay within an internal action—modelled in the component's SEFF.. 1, 9, 10

SEFF loop loops (like Java's `for`, `while` and `do-while` statement) which affect the calls a component makes to other components. Such loops are—contrary to loops that stay within an internal action—modelled in the component's SEFF.. 2, 9–11

service effect specification describes the inner behaviour of a component on a highly abstract level by specifying the relationship between provided and required services of a component. . 2

software architect a person who planes a software architecture from existing components and interfaces. Uses architectural styles and patterns, analyses architectural specifications, and makes design decisions. . 3

software architecture todo. 1, 2, 7

SoMoX a Palladio plugin for static code analysis to re-engineer a software's architecture from its source code. Constructs a PCM instance including the reconstructed components and their SEFF.. 1–3, 11, 13

system deployer models the resource environments and allocations of components to resources. Also sets up the resource environments, deploys components onto resources, and maintains the running system. . 3

Bibliography

[1]

[2] Palladio component model, 09 2015.

[3] André van Hoorn, Jan Waller, and Wilhelm Hasselbring. Kieker: A framework for application performance monitoring and dynamic software analysis. In *Proceedings of the 3rd joint ACM/SPEC International Conference on Performance Engineering (ICPE 2012)*, pages 247–248. ACM, April 2012.