Science of Computer Programming 105 (2015) 73–91

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Usage contracts: Offering immediate feedback on violations of structural source-code regularities

Angela Lozano [a], Kim Mens [a,*], Andy Kellens [b]

[a] Université catholique de Louvain, ICTEAM, Belgium
[b] Vrije Universiteit Brussel, Software Languages Lab, Belgium

ARTICLE INFO

Article history:
Received 8 August 2012
Received in revised form 28 August 2014
Accepted 19 January 2015
Available online 20 March 2015

Keywords:
Software development tool support
Structural regularities
Source code analysis
Internal domain-specific language
IDE integration

ABSTRACT

Developers often encode design knowledge through structural regularities such as API usage protocols, coding idioms and naming conventions. As these regularities express how the source code should be structured, they provide vital information for developers using or extending that code. Adherence to such regularities tends to deteriorate over time because they are not documented and checked explicitly. This paper introduces *uContracts*, an internal DSL to codify and verify such regularities as 'usage contracts'. Our DSL aims at covering most common usage regularities, while still providing a means to express less common ones. Common regularities are identified based on regularities supported by existing approaches to detect bugs or suggest missing code fragments, techniques that mine for structural regularities, as well as on the analysis of an open-source project. We validate our DSL by documenting the structural regularities of an industrial case study, and analyse how useful the information provided by checking these regularities is for the developers of that case study.

1. Introduction

Being able to document and preserve architectural integrity and design knowledge of an application is important to increase its longevity [3]. Given that over time the actual implementation structure tends to drift away from initial architecture and design documents, programmers turn to structural source code regularities, such as naming conventions, coding idioms and usage protocols to embed design knowledge in the implementation. Complying with these structural regularities facilitates future changes as they convey coding and design assumptions that need to be respected for the program to remain well designed and correct. In practice however, these regularities often get violated simply because they are not encoded or checked explicitly. Systematic violation of structural regularities can lead to several problems, such as premature aging of the application or the introduction of defects. Matsumura et al. [29] report on a study in which they found that 32.7% of all bugs in a legacy system were caused by violations of implicit structural regularities.

In this paper we present *uContracts*, a tool for declaring structural source-code regularities (like API usage idioms and coding conventions) in a concise, explicit and verifiable way. The tool is conceived as a domain-specific language (DSL), embedded in the host language and IDE. The proposed DSL allows us to define low-level coding and implementation regularities, in terms of which higher-level regularities related to architecture, design or framework structure can be expressed.

* Corresponding author.
E-mail addresses: angela.lozano@uclouvain.be (A. Lozano), kim.mens@uclouvain.be (K. Mens), andy.kellens@vub.ac.be (A. Kellens).

http://dx.doi.org/10.1016/j.scico.2015.01.004
0167-6423/

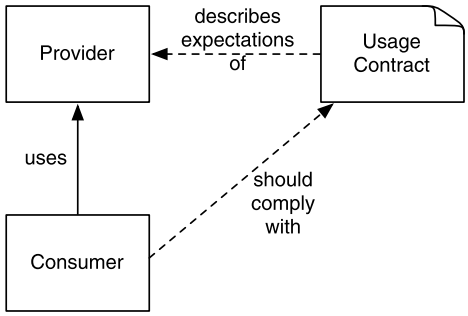

Fig. 1. A usage contract, depicted as a contract between a provider and a consumer.

We use the generic term *structural source-code regularity* for any of the patterns that can be described in our DSL, since the DSL is not limited to expressing architectural, design or framework patterns, but can also express more low-level coding idioms.

Our *uContracts* DSL was prototyped[1] in the Pharo Smalltalk development environment, because of Smalltalk's facilities for prototyping such tools. Nevertheless, the idea behind the tool remains essentially independent of the language and will therefore be presented as such in the paper.

As illustrated schematically in Fig. 1, in the *uContracts* DSL, structural regularities are expressed as *usage contracts* between two parties: a *provider*, i.e. the code entities that will be (re)used, and a *consumer*, i.e. the code entities that will use or extend the (re)used entities. A usage contract defines the expectations of and assumptions made by the reusable entities on the entities that reuse it. Consider for example an application or framework implementing some graphical editors. To maintain a consistent behavior among the classes implementing these editors, they should respect a variety of implementation guidelines such as: extending the default editor instead of the abstract editor, complying with certain naming conventions, being labeled with appropriate annotations, implementing the command pattern (i.e., providing an `isUndoable` method, and an `undo` method when necessary, and following a certain implementation template), etc. It is possible to define a usage contract that describes such regularities and to verify that the classes that implement such editors comply with these regularities.

The idea and terminology of declaring the assumptions that reusing code can make about the code it reuses as a *contract* between two parties, is loosely inspired by our previous work on *reuse contracts* [41]. The underlying approach is different however. The main purpose of the internal DSL presented in the current paper is to offer developers the necessary primitives – similar to what unit testing frameworks do – to express structural code regularities in a straightforward way, while remaining as close as possible to the syntax of the host language. Embracing the Pareto principle (a.k.a. the 80–20 rule), rather than offering a full language that allows us to express any possible regularity, we offer a simple and reduced set of language constructs that is sufficiently powerful to support a majority of frequently occurring structural regularities. Our DSL is complemented with tool support that, after each change to the source code, verifies automatically whether the source code still respects the encoded structural regularities and that provides fine-grained feedback regarding potential violations.

Our motivation for proposing an *internal* DSL to express structural regularities stems from our prior experience with developing and using an *external* DSL for that purpose. More specifically, in our earlier work on SOUL [8,32] and the IntensiVE tool suite [4,31,33] we explored how to express structural regularities in a declarative program query language on top of an object-oriented host language. In spite of the good symbiosis of the declarative language and supporting toolsuite with the underlying language and its IDE, the fact that developers had to learn a new declarative language in which to express their regularities, turned out to be a major inhibitor towards adoption. This prior experience also taught us that, while having a Turing-complete specification language allowed for great flexibility, in practice describing most regularities required only the use of a small subset of the features of our language. We thus decided to develop instead a more lightweight and limited language as an internal DSL, in order for the developer to remain in his comfort zone when encoding these regularities.

The current paper introduces this DSL and presents the following contributions:

- The definition of the *uContracts* DSL for defining usage contracts that capture the structural regularities governing a software system;
- An argumentation that the *uContracts* DSL covers many common regularities, based on a thorough analysis of different kinds of commonly occurring regularities, a literature study and an investigation of the JHotDraw system;
- Prototype tool support in terms of an integration with the Smalltalk language and Pharo IDE;
- A first industrial case study to assess the usefulness of declaring and checking structural regularities by means of usage contracts.

[1] http://www.squeaksource.com/eContracts.html.