The Journal of Systems and Software 118 (2016) 234–250

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Evaluating refactorings for spreadsheet models

Jácome Cunha[a], João Paulo Fernandes[b], Pedro Martins[c], Jorge Mendes[c], Rui Pereira[c,*], João Saraiva[c]

[a] NOVA LINCS, DI, FCT, Universidade Nova de Lisboa, Portugal
[b] LISP - RELEASE, Universidade da Beira Interior & HASLab/INESC TEC, Portugal
[c] Universidade do Minho & HASLab/INESC TEC, Portugal

A R T I C L E   I N F O

Article history:
Received 1 June 2015
Revised 18 December 2015
Accepted 18 April 2016
Available online 3 May 2016

Keywords:
Software refactoring
Model-driven engineering
Spreadsheets
Empirical study,

A B S T R A C T

Software refactoring is a well-known technique that provides transformations on software artifacts with the aim of improving their overall quality.

We have previously proposed a catalog of refactorings for spreadsheet models expressed in the ClassSheets modeling language, which allows us to specify the business logic of a spreadsheet in an object-oriented fashion.

Reasoning about spreadsheets at the model level enhances a model-driven spreadsheet environment where a ClassSheet model and its conforming instance (spreadsheet data) automatically co-evolves after applying a refactoring at the model level. Research motivation was to improve the model and its conforming instance: the spreadsheet data.

In this paper we define such refactorings using previously proposed evolution steps for models and instances.

We also present an empirical study we designed and conducted in order to confirm our original intuition that these refactorings have a positive impact on end-user productivity, both in terms of effectiveness and efficiency.

The results are not only presented in terms of productivity changes between refactored and non-refactored scenarios, but also the overall user satisfaction, relevance, and experience.

In almost all cases the refactorings improved end-users productivity. Moreover, in most cases users were more engaged with the refactored version of the spreadsheets they worked with.

1. Introduction

Software refactoring (Fowler, 1999) is the process of modifying the source code of software programs without changing their semantics. That is to say that while improvements are expected on the non-functional attributes of a piece of software, it is mandatory that its associated functional attributes are not affected by refactorings.

Improvements can be achieved, for example, by transforming the software into a new version with reduced complexity, or with added expressiveness in either the code or its model (or both), or with diminished overall size (fewer methods, classes, or lines of code).

In practice, a significant set of automated refactorings is usually available for a concrete programming language. This reduces the overall programming effort, since due to the improved quality of refactored code traditional programming tasks become simpler and can be implemented faster (Fowler, 1999).

Because of its generic applicability, code refactoring has been studied in different contexts, ranging from software source code (Fowler, 1999; Mens and Tourwe, 2004) or software models (Einarsson and Neukirchen, 2012), to spreadsheets (Badame and Dig, 2012). We have ourselves proposed (Cunha et al., 2014) a series of refactorings for ClassSheets (Engels and Erwig, 2005).

ClassSheets are a high level, object-oriented modeling language for spreadsheets. Integrating concepts from the Unified Modeling Language (UML), this language provides a modular and abstract

* Corresponding author.
E-mail addresses: jacome@fct.unl.pt (J. Cunha), jpf@di.ubi.pt (J. Paulo Fernandes), prmartins@di.uminho.pt (P. Martins), jorgemendes@di.uminho.pt (J. Mendes), ruipereira@di.uminho.pt (R. Pereira), jas@di.uminho.pt (J. Saraiva).

http://dx.doi.org/10.1016/j.jss.2016.04.043
0164-1212/© 2016 Elsevier Inc. All rights reserved.

methodology for dealing with spreadsheets, and namely to specify and maintain their business logic.

The appearance of this modeling language allowed us to develop an environment where concrete spreadsheets (or spreadsheet instances) are automatically derived from, and maintained together with abstract specifications (or spreadsheet models) (Engels and Erwig, 2005; Cunha et al., 2012c). This means that an evolution step on either the spreadsheet instance or its model is automatically propagated to the associated artifact, ensuring their consistency at all times (Cunha et al., 2012b).

In such a model-driven setting, we have shown (Cunha et al., 2015) that end-users are more efficient (that is, they complete equivalent tasks in less time) and effective (that is, they commit less errors) when they use a model-driven spreadsheet over regular spreadsheet data. In fact, that paper presents an empirical study showing that errors can be prevented by carefully reasoning about, and designing, a concise model, instead of doing so with a potentially large spreadsheet. This confirms an earlier idea that different, more refined, representations (or models, even though these representations are all at what we consider here the spreadsheet instance level) for data in a spreadsheet can improve productivity of end users (Beckwith et al., 2011).

In this paper we revise the refactorings proposed in Cunha et al. (2014) with the goal of improving the overall quality characteristics of ClassSheet models. The proposed refactorings are: *extract class, inline class, move attribute, move formula*, and *remove middle man*.[1] Moreover, we have specified these refactorings using our bidirectional transformational system previously introduced in Cunha et al. (2012b). This is the first contribution of this paper.

Later, we also assess in practice the refactorings catalog of (Cunha et al., 2014). With our work, we seek to find the answers to the following research questions:

(i) Do the spreadsheet instances (automatically) derived from refactored ClassSheet models allow end users to be more *efficient* than they would be if manipulating the instances derived from the corresponding original (non-refactored) models?
(ii) Do the spreadsheet instances derived from refactored ClassSheet models allow end users to be more *effective* than they would be if manipulating the instances derived from the corresponding original models?
(iii) Do spreadsheet users have a better experience working with the spreadsheet instances derived from refactored ClassSheet models instead of working with the instances derived from the corresponding original models?

That is to say that we propose to evaluate ClassSheet refactored models by analyzing the productivity in their instances, since these are standard spreadsheets, a software artifact that is used daily by millions of end users worldwide. We analyze the effectiveness and efficiency aspects of productivity, and also by measuring the overall experience of their users. With this goal in mind, we have designed and conducted an empirical study with spreadsheet end users, being this study, as well as the analysis of its results, the second and main contribution of this paper. In this study, we analyzed the quantitative and qualitative differences of the usage of (already) refactored spreadsheet models versus non-refactored spreadsheet models from an end-user's perspective.

The lessons learned from the results of our study are very promising. Through a series of statistical experiments, we found evidence that refactorings do allow improvements of either the efficiency or the effectiveness of its instances (or both), with the exception of a single refactoring from the refactoring catalog proposed in (Cunha et al., 2014).

Finally, we also gathered from the participants of our study feedback of a more qualitative nature. While the analysis of such feedback is exposed to a certain degree of subjectiveness, we believe that most of it provides further evidence that the refactorings of Cunha et al. (2014) allow the improvement of other spreadsheet characteristics such as readability, understandability or overall user satisfaction.

This paper is organized as follows. We start by introducing in Section 2 what model-driven spreadsheets are. In Section 3 we revise the previously introduced refactorings. In Section 4 we present the empirical validation of the refactorings proposed. Related work is presented in Section 5 and conclusions and future work in Section 6.

2. Model-driven spreadsheets

Engels and Erwig (2005)introduced the language ClassSheet to leverage handling spreadsheets to a more conceptual level. In a model-driven setting, a ClassSheet is the model and the spreadsheet data the corresponding instance. Indeed ClassSheets are more abstract than spreadsheets themselves, smaller, and easier to reason about. This language, which has a textual and a visual/graphical representation, has been embedded in spreadsheets themselves (Cunha et al., 2011). In such embedding the visual representation was used. Fig. 1 shows an embedded ClassSheet model of a small warehouse for a bar/coffee shop distribution (the numbered areas will be referenced in Section 3). Note this spreadsheet does not represent the actual data as it is shown in a second spreadsheets in Fig. 2.

On the top half (rows 1 through 6), we have three classes: **Product, Client**, and **Order. Product** (cell range `A3:B5` and `J3:K5`) contains a product `ID`, its `Name`, `Unit Price`, and amount in `Stock`, while expanding vertically (indicated by the ellipsis on row 5). **Client** (cell range `C1:G2`) contains the client's `Name`, along with his/her `Address`, `City`, and `Country`, and expands horizontally (indicated by the ellipsis on column `I`). The **Order** (cell range `C3:H5`) is a relationship class which arises due to the joining of a **Product** and a **Client**. This class contains a `Quantity` value of the product, an `Order Date`, a product `Category`, a `Sold Price` formula to calculate the price, and the warehouse's `ToSellprice` (expected price) for selling all of that product. The `ID` in the **Client** class references their **Contact Info**, a class on the bottom half (cell range `F8:H11`), which has the client's `Telephone` and `Email`. The `Seller`'s ID in the **Order** class references the **Seller** class (cell range `A8:B11`) which references the **SellInf** class (cell range `C8:E11`) containing the `Name`, `Cell` number, and `Home` number of the seller. These last three classes expand vertically.

Fig. 2 illustrates an instance of the model from Fig. 1. Starting from the bottom left corner, in a counter-clockwise direction, we can see instances for the `Seller`, `SellInf`, `ContactInf`, `Order`, two instances of `Client` (with the names Tiago C. and Marco C.) and four instances of `Product`.

Using ClassSheets we have created MDSheet (Cunha et al., 2012c), a framework that provides a bidirectional model-driven spreadsheet environment. The techniques and language described in that work allow transformations/evolutions from models to be automatically applied to the corresponding instances and vice-versa, as illustrated in Fig. 3.

Given a spreadsheet conforming to a ClassSheet, the user can evolve the model through an operation of the set Op_M, or the instance through an operation of Op_D. The performed operation on the model/instance is then automatically transformed into the corresponding set of operations on the instance/model using the *to*

[1] As the names suggest, and since ClassSheets resemble the object-oriented (OO) paradigm, the refactorings we proposed are based on the ones available in the OO realm.