Framework Instantiation Using Cookbooks Constructed With Static and Dynamic Analysis

Raquel F. Q. Lafetá and Marcelo A. Maia
Federal University of Uberlândia
Uberlândia, Brazil
Email: raquel.rafialho@gmail.com
marcelo.maia@ufu.br

David Röthlisberger
School of Informatics and Telecommunications
Faculty of Engineering
Universidad Diego Portales Santiago, Chile
Email: davidroe@mail.udp.cl

***Abstract*—Software reuse is one of the major goals in software engineering. Frameworks promote the reuse of not only individual building blocks, but also of system design. However, framework instantiation requires a substantial understanding effort. High-quality documentation is essential to minimize this effort. However, in most cases, appropriate documentation does not exist or is not updated. Our hypothesis is that the framework code itself and existing instantiations can serve as a guide for new instantiations. The challenge is that users still have to read large portions of code, which hinders the understanding process, thus our goal is to provide relevant information for framework instantiation with static and dynamic analysis of the framework and pre-existing instantiations. The final documentation is presented in a cookbook style, where recipes are composed of programming tasks and information about hotspots related to a feature instantiation. We conducted two preliminary experiments, the first to evaluate the recall of the approach and the second to study the practical usefulness of the recipe information for developers. Results reveal that our approach discloses accurate and relevant information about classes and methods used for framework instantiation.**

I. Introduction

Frameworks provide means to reuse existing design and functionality, but first require developers to understand how to use them. To support a wide range of specific applications, frameworks are designed with flexibility in mind, which leads to more abstract interfaces and a more complex understanding process. An object-oriented framework defines a set of permanent features known as frozen-spots to provide immutable features [1]. It also uses typical techniques of object orientation (e.g., subclassing and method overriding) to incorporate flexible features, the hotspots, which must be used appropriately to fulfill the needs of specific applications [2]. Each hotspot framework should be expanded with specific information about the application that is mainly known by application developers at the time of implementation [3].

In-depth comprehension of the framework design requires time. Understanding a single class without understanding its collaboration context has limited usefulness [4]. High-quality documentation is essential to simplify and guide the required understanding process. However, some frameworks have little documentation other than the source code and a set of examples [5]. The creation of high quality documentation for white-box frameworks is challenging [6], especially given the complexity of modern frameworks [7]. Frameworks that rely on inheritance usually require more knowledge from developers, and usually are called white-box frameworks [5], because the developer has complete access to the framework source code. They can understand the implementation details of the framework and can modify and extend parts of the framework in their customizations.

According to Johnson [5], the first use of a framework is always a learning experience. Often the best way to learn the range of applicability of a framework is by example, which is another reason why frameworks should encompass a rich set of examples [5]. Even when framework instantiation documentation such as tutorials is available, developers often want to look at real implementation examples for concrete guidance. Clear and working examples are an important complement to textual descriptions. When developers want to look at examples of framework instantiations, they can generally find such examples in existing applications [8]. Unfortunately, analyzing even a small set of examples manually is a laborious and error-prone process [9], given the large size of typical frameworks, the complexity of their interfaces, and the large number of possible instantiations.

To address this problem, we propose semi-automated construction of cookbooks for framework instantiation based on the source code of the framework itself and of existing instantiations that use the same framework version. We apply reverse engineering techniques such as dynamic analysis (feature location), static analysis and design pattern detection to retrieve information about the hotspots (classes and methods) used to instantiate framework features. The core idea is that the necessary instructions (hotspots to extend, methods to invoke, etc.) to instantiate a framework can be reverse-engineered. The set of instructions is presented in a cookbook-like document composed of recipes, along with concrete reverse-engineered examples of instantiations of one particular feature [10].

We have conducted two preliminary experiments evaluating whether recipes provide the correct and appropriate information required by developers for concrete framework instantiations; results reveal that our cookbook approach is able to efficiently drive new instantiations.

We present in Section II a semi-automated approach to generate a cookbook of recipes that drive the framework instantiation activity. We conducted two preliminary user studies to explore the achieved recall of our approach and its practicality, presented in Section III. Section IV explores related work and Section V discusses the approach and concludes the paper.

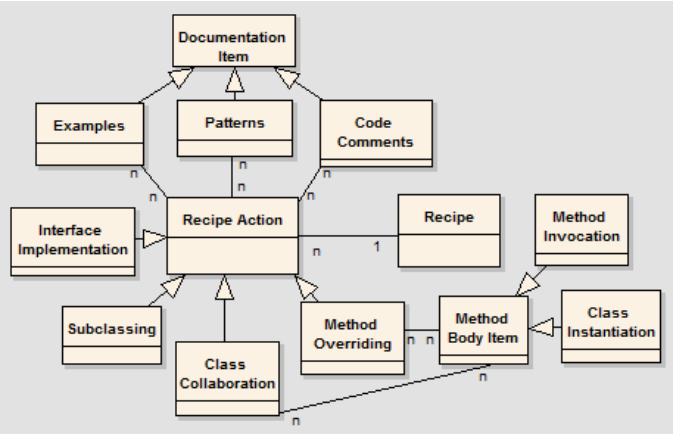

Fig. 1: Cookbook Meta-model

II. Approach

A cookbook-like documentation approach can ease the framework instantiation process [5]. Cookbook is one of the most prominent examples of tools that provide semi-automated assistance for the framework instantiation process [11], [5].

Our cookbook contains feature recipes, that is, a list of tasks to perform for instantiating a particular framework feature, along with concrete examples illustrating these tasks. Beginner programmers can use a cookbook to realize their first application, more advanced programmers look up solutions to particular problems.

Cookbooks have a semi-structured content based on chapters and recipes, which also have a typical structure [12], [11]. Figure 1 presents the meta-model that serves as a formal definition of our cookbook. A cookbook is composed of several recipes, one for each desired feature of the instantiated application. The recipes are composed of tasks or actions to implement the feature. Three main tasks exist: (i) Subclassing, to extend a hotspot, presented as a index of hotspot classes that can be extended; (ii) interface implementation, to implement a hotspot interface, presented as an index; and (iii) class collaboration, that is, how to collaborate between application and hotspot classes, displayed as an index of hotspot classes that can be used. These tasks are enriched with information about hotspot classes: Full class declaration; design patterns to be applied; code comments; and a list of methods from the hotspot class to override and/or implement. For each method, concrete usage examples, code comments, and their role in the applied design pattern are presented. Table I presents an extract of the information for the recipe *Create New Figure*.

Four phases are necessary to construct the recipes:

1) Extract Dynamic Information Phase: Finding a feature such as implementing an editor within an existing application requires identifying those pieces of code that make up that extension. This can be challenging because those pieces of code might not be located together and be intermixed with other code, and some pieces of code might be shared among several extensions [8]. To deal with this problem, we use feature-location techniques based on dynamic analysis [13], that is, trace extraction is used to locate the sets of code elements implementing the features of interest in existing applications. Those code elements (classes and methods) are obtained from feature trace execution. These traces are extracted during the execution of a scenario that represents adequately the feature. Hence it is crucial to study appropriate usage scenarios to obtain a complete and accurate coverage of a framework feature. During the extraction process, trace files are created to capture method-call events generated during execution.

A new framework user or an expert can use this approach for their own purpose or to distribute it as a framework document. The approach obtains the information in a structured and systematic way, instead of looking for information at the code and manually implementing a document. We believe that the effort to document a framework can be minimized using our approach, which we will evaluate in future work.

2) Extract Static Information Phase: Static information about the framework and existing application source code are extracted in this phase. Rigi Standard Format[1] and Crocopat[2] are used to obtain static information and relationships from the framework and instantiation examples. We use the Crocopat tool to create rules and identify the hotspots, frozen-spots, subclasses, interfaces, overridden methods, class collaborations, and method invocations. Also, at this stage we obtain comments and examples of instantiations from the source code. Design patterns used in the hotspots are obtained using an approach created by Tsantalis and colleagues [14]. Their methodology fully automates the pattern detection process by extracting the actual instances in a system for the patterns that the user is interested in. Johnson [5] suggests documenting applied design patterns as a means for programmers to understand frameworks. Design patterns lead to the construction of well-structured, maintainable, and reusable software systems and capture the intent behind a design by identifying objects, their collaborations and the distribution of responsibilities.

3) Filter Phase: In the previous stages we obtained the hotspots' static and dynamic information. In this phase, a filter is performed on the static information to focus only on the information about the hotspots present in feature traces (dynamic information). The trace decides about the information to be included in the feature recipe.

4) Create Recipe Phase: At this stage we generate the recipes with the information obtained in the previous steps. Currently, the recipe is structured as a static document, but we intend to create a dynamic document that adapts itself according to a developer's preferences and actions [11].

To create an application using a framework, the developer must know what is possible to build with it. Our cookbook presents an index of recipes to implement features that can be realized with the framework, this index lists features and provides a description of those. Recipes are structured by features because the developer usually starts the development process from a list of features that must be implemented. When the developer chooses a feature to implement, she needs to know which hotspots to use and how, that is, understanding the interactions within the framework. To address this problem, recipes presents an index of hotspots that could be used to

[1] http://www.rigi.cs.uvic.ca/downloads/rigi/doc/node52.html
[2] https://code.google.com/p/crocopat/

implement a given feature and present tasks to actually implement these hotspots, based on existing application samples. This index shows the description of each hotspot. This information is obtained from the hotspot code comments. The tasks present information to guide developers during the instantiation process, such as code comments for hotspot classes, design patterns applied in the hotspot, methods to override and their comments, and usage samples for these methods based on code of existing applications.

TABLE I: Extract of recipe *Create New Figure* for the subclassing task *Extends StandardDrawing*

Task and hotspot:	Extends StandardDrawing
Code comments (class):	The standard implementation of the Drawing interface. @see Drawing
Full hotspot class declaration:	public class StandardDrawing extends CompositeFigure implements Drawing
Design Pattern (Class):	Design Patterm: Observer Subject: ch.ifa.draw.standard.StandardDrawing observers: Vector fListeners. Attach(Observer):addDrawingChangeListener. Detach(Observer): removeDrawingChangeListener. Notify(): figureInvalidated, figureRequestUpdate Observer: ch.ifa.draw.framework.DrawingChangeListener Update(): drawingInvalidated, drawingRequestUpdate
Hotspot methods:	remove(Figure figure)
Comments (methods):	Removes the figure from the drawing and releases it.

Usage examples (methods):	public synchronized Figure remove(Figure figure) {
	Figure f = super.remove(figure);
	if (f instanceof AnimationDecorator)
	return ((AnimationDecorator) f).peelDecoration();
	return f; }

III. Preliminary Evaluation

We perform two preliminary studies with human subjects, one to assess the recall rate of hotspots in recipes and the second to evaluate if and how useful the presented recipe information is perceived in a concrete instantiation activity.

A. Cookbook Recall

With the first preliminary study, we aim at evaluating if recipes contain the hotspots required for the framework instantiation (recall). We instructed eight groups, each including two graduate students in C.S. from the Federal University of Uberlândia, to instantiate the framework JHotDraw 5.3[3]. Each group developed a system named JHotDER which draws entity relationship diagrams. To generate the recipes, we used the existing applications JHotDraw APP and JModeller.

TABLE II: Recall study: Hotspot classes used in JHotDER solutions and their presence in recipes.

Hotspots used in solutions	% Presence in solutions	Present in recipes?	Is hotspot in samples?	Present in traces?
AttributeFigure	10%	no	no	yes
ConnectionFigure	10%	no	yes	no
LineConnection	100%	yes	yes	yes
LineFigure	100%	yes	yes	yes
MDLDrawApplication	100%	yes	yes	yes
PolyLineFigure	10%	yes	yes	yes
AbstractLineDecoration	70%	yes	yes	yes
CustomSelectionTool	100%	no	yes	no
GraphicalCompositeFigure	100%	yes	yes	yes

[3]http://www.jhotdraw.org

To evaluate the recall, we analyzed all implemented classes and all used hotspots in the eight JHotDER solutions provided by the different groups. We verified the presence of those hotspots in the recipe and obtained a recall rate of 66% for hotspot classes. As shown in Table II, nine hotspot classes have been used in the eight different JHotDER solutions. Three hotspots (AttributeFigure, ConnectionFigure and CustomSelectionTool) were not present in any recipe, meaning the cookbook would not help developers to instantiate these hotspots. The absence of these hotspots in the cookbook can have two reasons (i) a class was not executed in any feature trace and (ii) a class is not a hotspot, but a frozen-spot. AttributeFigure is not present in recipes because it is a frozen-spot, misused by students. The ConnectionFigure and CustomSelectionTool, true hotspots, have not been executed in any execution trace used to generate the cookbook. Feature location approaches usually suffer from coverage problems [15], [13].

About the recall of hotspot methods in recipes, students used 41 hotspots methods, 21 of those are present in recipes, which represents a recall rate of 52%. The three hotspot classes not included in recipes are responsible for 48% of the absent methods, since the approach just lists hotspot methods of classes included in recipes.

B. Usefulness of Recipes

TABLE III: Study for the usefulness of recipe information

Subjects	H-C	FD-C	DP-C	CC-C	H-M	CC-M	DP-M	CE-M	M
Subject 1	1	2	1	-1	2	2	1	2	-2
Subject 2	1	1	-1	0	-2	-2	-1	0	-2
Subject 3	1	1	2	1	1	1	2	1	-1
Subject 4	1	-1	-2	-2	-2	-2	-2	2	-2
Subject 5	1	2	1	2	1	2	2	2	-1

We conducted a second study with five graduated software developers with experience using frameworks. We asked them to realize a framework instantiation for creating a new feature in JHotDraw 5.3, namely implementing a hexagon figure.

The participants received three different documents to support them in this activity: A document with instructions and the task description, the recipe related to drawing a new figure in JHotDraw 5.3 in a textual format, and a questionnaire. The questionnaire contained questions about the usefulness of recipe information and also included an artificial control question to test for bogus answers in order to verify if the participants adequately answered the questions. This control question was about usefulness of metrics from the framework.

All participants performed the activity successfully and completed within the three hours time limit. They performed the activity with an average time of 1 hour and 32 minutes. The subjects evaluated the recipe information as presented in Table III, following a Likert scale from -2 to +2, with the following meaning of the different ratings: (+2) *Heavily Useful*: Information considered to be as very useful and complete (the participant did not need external information, such as having to look in the code to understand what should be done); (+1) *Weakly Useful*: Information is considered to be useful but not complete; (0) *Neutral View*: Information neither useful

nor useless; (-1) *Weakly Useless*: The information was not very useful for the subject; (-2) *Strongly Useless*: The subject considered the information as completely useless. The names of recipe information are abbreviated in the Table III as follows: *H-C*: Hotspot (classes) index; *FD-C*: Full hotspot class declaration; *DP-C*: Design patterns (classes); *CC-C*: Code comments (classes); *H-M*: Hotspot methods; *CC-M*: Code comments (methods); *DP-M*: Design patterns (methods); *CE-M*: Code examples (methods); *M*: Metrics (control question).

The majority of subjects considered to be useful (Likert scale 2 and 1) the recipe information with the exception of code reviews for hotspot classes that had a neutral classification, having one vote for each classification level. Hotspot (classes) index, Full hotspot class declaration and Code examples presented the most significant results with, respectively, 5 out 5, 4 out 5 and 4 out 5 of the participants classifying those as useful. For this experiment, all hotspots used in the activity were described in the recipe, representing a recall rate of 100%.

The last question of the questionnaire was a qualitative question asking about the general perception of recipe usefulness. All participants reported that recipes assisted them during the instantiation process. The majority demonstrated satisfaction in using the recipes, but they suggested improvements in the presentation of the information, justified by the large amount of information shown in one recipe. We plan to address this important observation by providing an interactive tool to quickly search and filter the recipe information.

IV. Related Work

A wide variety of software documentation techniques have been proposed to support framework understanding and usage. Krasner et al. [12] built a cookbook for Model-View-Controller frameworks, that describes the framework textually and presents code examples. However, the recipes do not follow a consistent structure. In [11], authors present SmartBooks, a cookbook with an interactive interface. It was created based on instantiation rules provided by experts for the framework. [16] presents Design Fragments, a description of what the programmer must instantiate to use the framework. [17] presents ReuseTool to orchestrate tasks within a process of reuse specified by a framework expert using RDL (Reuse Description Language). These existing approaches use specialist knowledge to generate information. [9] describes the Guido tool for exploring source code examples, using multiple coordinated views to visualize the relationships between examples.

V. Conclusions

The first preliminary experiment was useful to evaluate if the approach is able to generate the recipes with reasonable recall. This recall rate is one of our concerns which we intend to address in future work. The study's qualitative results show that the majority of the recipe information was considered to be useful in a concrete framework instantiation activity. These results motivate us to maintain the current recipe information but improve its presentation in future work. In particular we aim to reduce the required effort of developers during the instantiation activity and to improve comprehension of the hotspot instantiation. Currently, the process of reverse engineering has been defined and tested and is being automated with third-party tools. We intend to conduct more extended qualitative surveys in order to ensure whether recipes are complete, accurate and useful for developers. To evaluate whether our cookbook reduces the effort for framework instantiation and comprehension, we also plan to perform controlled experiments with human subjects.

Acknowledgment

We gratefully acknowledge the financial support of CONICYT and CAPES for the STIC-AmSud project "Mining software repositories to instantiate software frameworks and react to API changes". D. Röthlisberger is also partially funded by FONDECYT Project 1140068.

References

[1] W. Pree, *Design patterns for object-oriented software development*. NY, USA: ACM Press/Addison-Wesley Publishing Co., 1995.

[2] M. E. Markiewicz and C. J. P. de Lucena, "Object oriented framework development," *Crossroads*, vol. 7, no. 4, pp. 3–9, jul 2001.

[3] D. Georgakopoulos, M. Hornick, and A. Sheth, "An overview of workflow management: From process modeling to workflow automation infrastructure," *Distrib. Parallel Databases*, vol. 3, no. 2, pp. 119–153, apr 1995.

[4] M. Bruch, T. Schäfer, and M. Mezini, "Fruit: Ide support for framework understanding," in *Proc. of the 2006 OOPSLA Workshop on Eclipse Technology eXchange*. NY, USA: ACM, 2006, pp. 55–59.

[5] R. E. Johnson, "Components, frameworks, patterns," in *Proc. of the 1997 Symposium on Software Reusability*, ser. SSR '97. New York, NY, USA: ACM, 1997, pp. 10–17.

[6] J. Bloch, *Effective Java Programming Language Guide*. Mountain View, CA, USA: Sun Microsystems, Inc., 2001.

[7] D. Kirk, M. Roper, and M. Wood, "Identifying and addressing problems in object-oriented framework reuse," *Empirical Softw. Engg.*, vol. 12, no. 3, pp. 243–274, jun 2007.

[8] B. Dagenais and H. Ossher, "Automatically locating framework extension examples," in *Proc. of the 16th ACM SIGSOFT International Symposium on Foundations of Softw. Eng.*, ser. 08/FSE-16. New York, NY, USA: ACM, 2008, pp. 203–213.

[9] R. Cottrell, B. Goyette, R. Holmes, R. Walker, and J. Denzinger, "Compare and contrast: Visual exploration of source code examples," in *Visualizing Software for Understanding and Analysis, 2009. VISSOFT 2009. 5th IEEE International Workshop on*, Sept 2009, pp. 29–32.

[10] T. Eisenbarth, R. Koschke, and D. Simon, "Locating features in source code," *Soft. Eng., IEEE Trans. on*, vol. 29, no. 3, pp. 210–224, March 2003.

[11] A. Ortigosa and M. Campo, "Smartbooks: a step beyond active-cookbooks to aid in framework instantiation," in *Technology of Object-Oriented Languages and Systems, 1999. Proc. of*, 1999, pp. 131–140.

[12] G. E. Krasner and S. T. Pope, "A cookbook for using the model-view controller user interface paradigm in smalltalk-80," *J. Object Oriented Program.*, vol. 1, no. 3, pp. 26–49, aug 1988.

[13] M. de Almeida Maia and R. F. Lafetá, "On the impact of trace-based feature location in the performance of software maintainers," *J. Syst. Softw.*, vol. 86, no. 4, pp. 1023 – 1037, 2013.

[14] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, and S. Halkidis, "Design pattern detection using similarity scoring," *Soft. Eng., IEEE Trans. on*, vol. 32, no. 11, pp. 896–909, Nov 2006.

[15] B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, and R. Koschke, "A systematic survey of program comprehension through dynamic analysis," *Software Engineering, IEEE Transactions on*, vol. 35, no. 5, pp. 684–702, 2009.

[16] G. Fairbanks, D. Garlan, and W. Scherlis, "Design fragments make using frameworks easier," in *Proc. of the 21st Annual ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications*, ser. OOPSLA '06. NY, USA: ACM, 2006, pp. 75–88.

[17] T. C. Oliveira, P. Alencar, and D. Cowan, "Reusetool-an extensible tool support for object-oriented framework reuse," *J. Syst. Softw.*, vol. 84, no. 12, pp. 2234–2252, dec 2011.