International Journal of Software Engineering and Knowledge Engineering

An automated approach for constructing framework instantiation documentation

Raquel Fialho de Queiroz Lafetá
Faculty of Computing, Federal University of Uberlândia, Campus Santa Mônica
Uberlândia, MG, 38400-902, Brazil
raquel.rafialho@gmail.com

Thiago Fialho de Queiroz Lafetá
Faculty of Computing, Federal University of Uberlândia, Campus Santa Mônica
Uberlândia, MG, 38400-902, Brazil
fialhot@gmail.com

Marcelo de Almeida Maia
Faculty of Computing, Federal University of Uberlândia, Campus Santa Mônica
Uberlândia, MG, 38400-902, Brazil
marcelo.maia@ufu.br

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

A substantial effort, in general, is required for understanding APIs of application frameworks. High-quality API documentation may alleviate the effort, but the production of such documentation still poses a major challenge for modern frameworks. To facilitate the production of framework instantiation documentation, we hypothesize that the framework code itself and the code of existing instantiations provide useful information. However, given the size and complexity of existent code, automated approaches are required to assist the documentation production. Our goal is to assess an automated approach for constructing relevant documentation for framework instantiation based on source code analysis of the framework itself and of existing instantiations. The criterion for defining whether documentation is relevant would be to compare the documentation with an traditional framework documentation, considering the time spent and correctness during instantiation activities, information usefulness, complexity of the activity, navigation, satisfaction, information localization and clarity. We propose an automated approach for constructing relevant documentation for framework instantiation based on source code analysis of the framework itself and of existing instantiations. The proposed approach generates documentation in a cookbook style, where the recipes are programming activities using the necessary API elements driven by the framework features. We performed an empirical study, consisting of three experiments with 44 human subjects executing real framework instantiations aimed at comparing the use of the proposed cookbooks to traditional manual framework documentation (baseline). Our empirical assessment shows that the generated cookbooks performed better or, at least, with non-significant difference when compared to the traditional documentation, evidencing the effectiveness of the approach.

Keywords: Program comprehension, reverse engineering, documentation, frameworks, empirical assessment.

1. Introduction

Frameworks are widely used and are an effective form of software reuse [9]. Frameworks promote the reuse of individual building blocks, but also the reuse of system design. However, frameworks tend to have a higher learning curve compared to libraries due to the complexity caused when there is a large number of inter-related abstract classes, present in more sophisticated designs. Object-oriented frameworks use typical techniques of object orientation (e.g., subclassing and method overriding) to incorporate flexible features, the hot-spots, which must be extended appropriately to fulfill the needs of specific applications [20]. This situation is even more dramatic for white-box frameworks, which typically rely on inheritance, and requires understanding part of the framework source code for implementing the required extensions [14].

According to Johnson [14], [15] and Shull et al [28], the best way to learn about the applicability of a framework is by example, especially for developers starting to learn how to instantiate a framework [14]. Developers often want to look at real implementation examples for concrete guidance. This proposal is supported by the increasing availability of software repositories that provide source code that can serve as code examples [15], [12], [21]. Furthermore, modern object-oriented frameworks typically come with a number of demo applications, which developers can use to learn about hot-spots [11]. Unfortunately, analyzing a small set of examples manually is a laborious and error-prone process [4], given the large size of typical frameworks, the complexity of their interfaces, and the large number of other possible instantiations.

Well-structured documentation could help in guiding the instantiation process. However, some frameworks have little documentation other than the source code and a set of examples [11]. The creation of high quality documentation for frameworks is challenging [35], especially given the complexity of modern frameworks [16]. To address this problem, we propose an automated approach for the construction of structured documentation for framework instantiation based on the source code of the framework itself and of existing instantiations (code examples) that use the same framework version. So, framework developers could leverage on an automated approach to produce final high quality documentation for framework users, that could be more easily updated for new versions of the framework given an automated process.

We apply reverse engineering techniques and propose a hybrid approach using dynamic analysis (feature location), static analysis and design pattern detection [34] to retrieve information concerning the hot-spots (classes and methods) used to instantiate framework features. The core idea is that the necessary instructions to be deployed in the documentation, e.g., hot-spots to extend, methods to override, etc. can be reverse-engineered. The chosen representation of the documentation to organize the set of instructions is a cookbook-like document composed of recipes, along with concrete reverse-engineered examples of instantiations of one feature [7].

This work is an extension of a previous one [18] that presented preliminary results on the precision of the information and structure of a generated cookbook. Those results contributed to the definition of improvements in the information extraction, creation of a

new tool and changes in the cookbook presentation.

The main contribution of this paper is the construction of an automated approach for producing framework instantiation documentation for Java systems that has quality at least similar to traditional manual documentation considering the criteria of time spent and correctness during instantiation activities, information usefulness, complexity of the activity, navigation, satisfaction, information localization and clarity. Moreover, we demonstrate the goal of this work with an empirical study, consisting of three experiments with 44 human subjects executing real instantiations with two different frameworks.

The paper is organized as follows. In Section 2, we present some related works. In Section 3, we present the approach to generate a cookbook of recipes. In Section 4, we provide the details of the experimental setting. In Section 5, the results are presented, and discussed in Section 6. Finally, in Section 7, our conclusions and future work are presented.

2. Related Work

2.1. *Documentation for framework instantiation*

A wide variety of software documentation techniques have been proposed to support framework understanding and usage. Approaches based on examples have emerged motivated by the lack of specialist or appropriate documentation and the utility of instantiation examples. These approaches usually extract information from existing instantiations (source code). For example, Jiang and colleagues [13] propose an automated approach for re-documenting UML sequence diagrams for APIs. FRUIT is another tool created for this purpose, and combines the use of data mining techniques with a context-dependent presentation to extract reuse patterns from existing framework instantiations [2]. Users can then automatically query the rules that are relevant to the framework instantiations learning. These tools assist the developer in identifying similarities and differences in sets of examples.

There is some previous work that proposed the creation of software repositories which include examples of framework instantiations [22, 36, 12]. However, developers should learn a new query language [10] and have a pre-conceived idea of what kind of example might help them with their activities [21], or write the source code in a style that conforms to the available examples in the repository [36]. In order to mitigate this difficulty, Holmes and Murphy [12] implemented an approach along with the Strathcona tool, which is based on heuristically matching the structure of the code under development to the example code. Kim et al. [15] present a system to build a repository of candidate code examples by interrogating an existing code search engine; summarizing those candidate code examples and extracting semantic features from the summarized code snippets to finds the most representative code examples. Their user study with 24 students subjects show that this approach provides code examples with high precision and boosts programmer productivity. This work reinforces that good code examples can boosts programmer productivity.

However, the above approaches still require the developer to know at least the names of some framework hot-spots that could be used in the implementation. Those approaches become less useful if the developer has only a high-level idea of which hot-spots are needed for feature instantiation. Several approaches use dynamic analysis to handle the problem

of locating code elements used toward a desired characteristic implementation. Those approaches do not require in-depth knowledge of the system samples to obtain the code elements [8], [19]. With respect to frameworks, Heydarnoori et al proposed FUDA (Framework API Understanding through Dynamic Analysis) [11], which is the most similar work to ours. Nonetheless, FUDA does not use static information, relying only on dynamic information, and it also does not present the documentation in a cookbook-like structure.

TaskNavigator is a tool for automatically extracting tasks from software documentation by conceptualizing tasks as specific programming activities [33]. They conducted a study where six professional developers used TaskNavigator for two weeks. This work emphasizes that the use of tasks for a specific implementation may be a good solution. Other recent works are the APITasks tool to generate task oriented API learning guide from Stack Overflow threads and source code comments, organized by a hierarchical task list [37], and the SISE tool to automatically augment API documentation with sentences from Stack Overflow that are related to a particular API type and that provide insight not contained in the API documentation [32].

In this work, we were motivated to combine the several features of previous approaches: example-based content, static relationship (structural and design-pattern analysis) and feature-driven documentation organization (feature location with dynamic analysis).

2.2. *Cookbook assessment*

Although there is no framework to assess the quality of technical cookbooks, at the best of our knowledge, there has been already some work on assessing technical documents. Souza et al. [31] propose an approach to automatically generate cookbooks from the crowd knowledge available in Stack Overflow. To evaluate the quality of the generated cookbooks, they propose a set of criteria that are assessed in an empirical study with human subjects. More generally, Arthur and Stevens [1] developed a taxonomy for evaluating adequate documentation with four criteria: accuracy, completeness, usability, and expandability. Smart [29] studied quality factors for technical documentation in the context of software quality documents [29], and proposed three criteria to evaluate quality: easiness to use (i.e. task-oriented), easiness to understand (i.e. concrete including examples), and easiness to find (i.e. organized in a way that makes sense to the user). Later, Robillard [26] observed that the following properties of software documentation are the most important: content (information in the document), organization (index, sections, subsections), use of examples, and being up-to-date.

Considering the above criteria, we define a list of desirable properties of technical documentation, and how we address their respective assessment in the context of the generated cookbooks:

- *Usability*. This criteria is related to the easiness to use, to understand and to find from [29], and organization and use of examples from Robillard [26]. For this criteria, we propose an experimental study, where developers evaluate navigation, satisfaction, information localization and clarity.

- *Completeness*. This criteria is out of the scope of cookbooks as they typically do not target completeness. Cookbooks are not meant to be reference manuals. Their main purpose is to empower adopters of a framework with the directions on how to solve selected programming tasks. The comprehensiveness of the generated cookbooks in this paper are related with how the chosen applications implemented with the framework explore its hotspots.
- *Expandability*. This criteria is related with the *completeness*, since the cookbooks could be more complete, if they are expanded with new recipes. Our approach is designed to be expandable in the sense that if we find new applications that instantiate the framework within new scenarios, we can re-execute the generation approach considering the new scenarios and new recipes would be generated.
- *Quality of content*. The content of the generated cookbooks are provided from the **facts** extracted from the source code. In this case, except for possible bugs on the developed artifacts and possible inaccuracies of the third-party software used in the implementation of the generation approach, the content is actually a factual information, and by definition correct. Nonetheless, although correct, one important thing to evaluate the quality of content is how developers can leverage that information to actually complete their tasks. To measure that quality, we propose to measure how successful are the developers, in terms of time and correctness, during the use of the cookbooks.
- *Up-to-date documentation*. From this perspective, an automatic approach for generation of documentation promotes the quality of up-to-documents, since whenever new versions of the frameworks and respective applications are released, the approach could be re-executed to provide a new version of the cookbooks.

3. A hybrid approach for cookbook construction

In this section, firstly we present the proposed structure for cookbooks aimed at guiding framework instantiation, and then we present the respective hybrid approach that generates of cookbooks for framework instantiation.

3.1. *Structure of cookbooks*

The proposed cookbooks consist of recipes based on the framework features, that is, a list of activities for instantiating a particular framework feature, along with concrete examples illustrating these activities. Beginners can use a cookbook to guide their first contact with the framework, reading and following a recipe of interest. More advanced programmers can look up for solutions of particular problems.

Cookbooks have a semi-structured content based on recipes, which also have a typical structure [17], [23]. Figure 1 presents the meta-model that serves as a formal definition of the proposed cookbook (item A). A cookbook is composed of several recipes (item B), one for each desired feature of the instantiated application. The recipes are composed of activities for implementing the respective feature (item C). There are four main activities organized within an index for each one of them: (C1) implementation of hot-spot

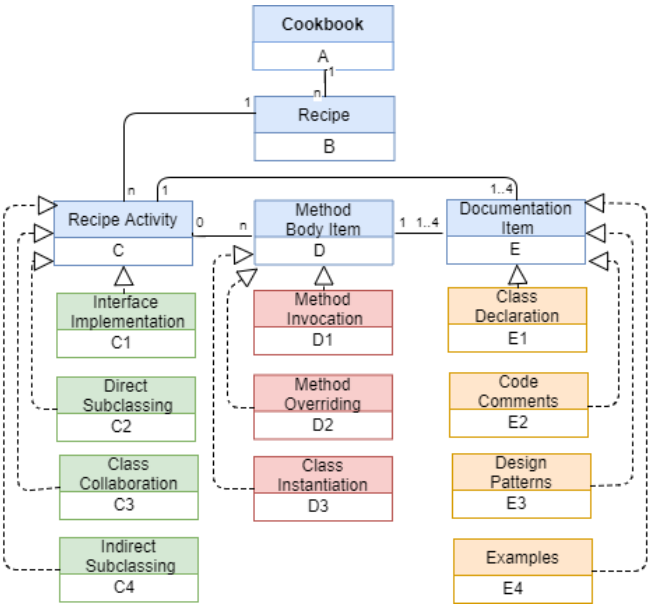

Fig. 1: The cookbook meta-model.

interfaces; (C2) direct subclassing of hot-spot classes; (C3) creation of collaborations with hot-spot classes; (C4) indirect subclassing of hot-spot classes by transitive inheritance.This information is complemented by activities that occur within methods (item D), such as: (D1) a list of methods from the hot-spot class to invoke; (D2) a list of methods from the hot-spot class to override; and/or (D3) instantiation of hot-spot class; These activities (C) and methods (D) are enriched with documentation items (item E) about hot-spot classes or methods: (E1) signature of one class declaration example that uses the hotspot; (E2) code comments on the hot-spot or; (E3) occurring design patterns; and, (E4) code examples of classes in existing instantiations that use those hot-spots classes or methods. An example of cookbook can be found at the project repository [a].

The cookbook could present other elements that would contribute to understanding. It is not our goal to evaluate which elements are best for composing this type of documentation. We define this minimum set of basic elements to understand a framework. Tests to evaluate if the items that make up the cookbook are really useful in the understanding process for using the framework were previously performed and reported in [18]. In addition, we expect that less polluted documentation could be better used. In the experiments presented in the next sections, we found that developers prefer more direct and clean documentation.

[a] `https://github.com/lascam-UFU/CookFrame`

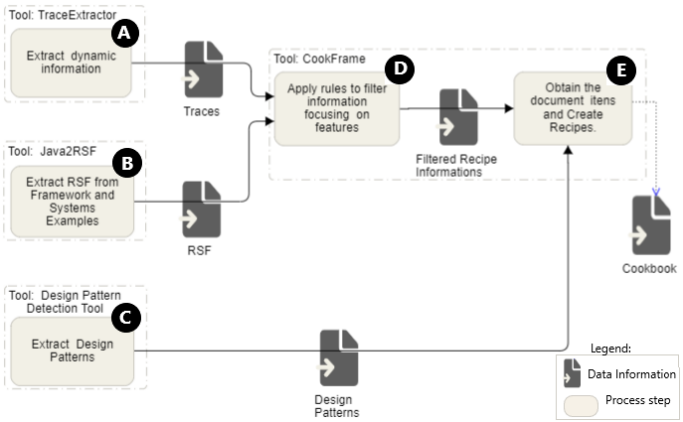

Fig. 2: The proposed approach for cookbook generation.

3.2. *Cookbook construction*

Firstly, the framework and examples of applications that will serve as the basis for the document creation are selected. The next four subsections will detail four phases of the automated process. Figure 2 summarizes this process. Algorithms 1 and 2 present a summary of the algorithm implemented to automate the creation of these cookbooks.

Algorithm 1 High level algorithm for cookbook construction

```
Input: sourceCodeDirectory, featureList, outputDocumentFile
1 init
2     traceTable.add(TRACEEXTRACTOR (sourceCodeDirectory);
3     rsfTable.add(JAVA2RSF (sourceCodeDirectory, featureList));
4     patternTable.add(DESIGNPATTERNDETECTIONTOOL (sourceCodeDirectory));
5     return COOKFRAME (traceTable, rsfTable, patternTable, sourceCodeDirectory, outputDocumentFile);
```

3.2.1. *Extract dynamic information*

The proposed cookbook structure presents the framework hot-spots and example of instantiations related to the list of features that will be documented. Locate this code can be challenging because those pieces of code may not be located together, may also be intermixed with other code and shared among several extensions [25] [5]. To handle this problem, we use a feature location technique based on dynamic analysis [19] where execution traces are used to locate the sets of code elements implementing the features of interest in existing applications. These traces are extracted during the execution of a scenario that represents adequately the feature. This step is represented in Figure 2 - A, and in Algorithm 1 as the function *TRACEEXTRACTOR*, which returns the method calls captured during the execution that are added to the *traceTable* .

We used an adapted version of the TraceExtractor tool presented at [19] to collect the

Algorithm 2 CookFrame system high level algorithm.

```
Input: traceTable, rsfTable, patternTable, sourceCodeDirectory, outputDocumentFile
Output: cookbookTable
init
    COOKFRAME (traceTable, rsfTable, patternTable, sourceCodeDirectory, outputDocumentFile)
        /* Get Recipe Activity */
        foreach specificClass X in rsfTable do
            foreach frameworkClass Y in rsfTable do
                /* Filter Rules shown in Table 1 */
                if (FilterRuleC1) then
                    recipeActivityTable.add (interfaceImplementatio(X,Y));
                if (FilterRuleC2) then
                    recipeActivityTable.add (directSubclassing(X,Y));
                if (FilterRuleC3) then
                    recipeActivityTable.add (classCollaboration (X,Y));
        /* Get Method Body Itens*/
        foreach specificMethod X.MethA(ParA) in rsfTable do
            foreach frameworkMethod Y.MethB(ParB) in rsfTable do
                /* Filter Rules shown in Table 1 */
                if (FilterRuleD1) then
                    methodBodyItensTable.add (methodInvocation(X.MethA(ParA), Y.MethB(ParB)));
                if (FilterRuleD2) then
                    methodBodyItensTable.add (methodOverriding(X.MethA(ParA), Y.MethB(ParB)));
                if (FilterRuleD3) then
                    methodBodyItensTable.add (classInstantiation(X.MethA(ParA), Y));
        /* Get Document Itens */
        foreach FrameworkClass hotspot in the recipeActivityTable do
            hotspotTable.add(getClassDeclaration(hotspot, sourceCodeDirectory));
            hotspotTable.add(getCodeComments(hotspot, sourceCodeDirectory));
            hotspotTable.add(getDesignPattern(hotspot, patternTable));
            hotspotTable.add.(getCodeExample(hotspot, sourceCodeDirectory));
        /*Generate the Cookbook in the HTML Format */
        foreach feature in traceTable do
            foreach hotspotFramework hotspot in the recipeActivityTable do
                if (hotspot ∈ feature) then
                    cookbookTable.getHotspotClassRecipeAction(feature, hotspot, recipeActivityTable, TraceTable);
                    cookbookTable.getHotspotMethodBodyItens(feature, hotspot, recipeActivityTable, TraceTable);
                    cookbookTable.getHotspotDocumentationItens(feature, hotspot, recipeActivityTable, TraceTable);
        writeFeatureHTMLDocument(cookbookTable, outputDocumentFile);
    return cookbookTable;
```

traces. The reasons for this choice are: i) the flexibility and simplicity of implementation; ii) the integration between the target system and the aspect responsible for capturing the information is transparent and can be easily enabled and disabled; and, iii) it is possible to locate the code without having to know the source code. The only need is to know the applications features and know how to execute them. A ”start” and ”end” markings of a feature execution enable the slicing of the differentiation of traces for features. The generated traces present a sequence of fully qualified method calls, creation of objects and also the stack level, which enables identifying the call tree.

A poorly defined and/or poorly executed trace collection can cause problems, such as collected code elements (method calls) that should not have been collected (false positives), and elements uncovered (false negatives). This type of problem, intrinsic to dynamic anal-

ysis, was recognized in several works [8]. However, our hybrid approach will use static analysis to verify the instantiation relationships in the code of hot-spots. This verification aims at minimizing the occurrence of false positives.

3.2.2. ***Extract static information***

Static information about the framework and existing application source code are obtained using the Java2RSF[b] tool. This tool returns the information in the Rigi Standard Format (RSF)[c], a suitable format for further processing. The RSF format present relations on *packages*, *classes*, *interface methods*, *parameters* and *types*. In addition, relationships between these elements are captured, such as, *extends*, *implements*, and *calls*. According to the packages referring to the code element, we also identify which classes are from the framework or which are from specific applications, example instantiations, defined with the relations FRAMEWORK or SPECIFIC, respectively included on the RSF result. It is possible identify the framework and application packages using the RSF from framework. This step is represented in Figure 2 - B, and also in Algorithm 1 by the function *JAVA2RSF*, which return the RSF relations that are added to the *rsfTable*.

In addition, a design pattern detection tool created by Tsantalis and colleagues [34] is used to obtain the design patterns related to the hot-spots and stored. This methodology fully automates the patten detection process by extracting the actual instances in a system for the patterns that the user is interested in. This approach achieved a good accuracy and precision with few false negatives and no false positives. This step is represented in Figure 2 - C, and in Algorithm 1 by the function *DESIGNPATTERNDETECTIONTOOL*, which returns the patterns that are added to the *patternTable*. After obtaining the hot spots of interest using the filtering rules in the next step, the tool can search the design patterns for these hot spots in the database.

3.2.3. ***Apply rules to filter information focusing on features***

In this phase, static information is filtered to focus only on the information concerning to framework hot-spots and specific classes of desired features, that is present on the feature traces. This step is represented in Figure 2 - D, and in Algorithm 2, in the IF commands guarded with the FilterRules. We developed the CookFrame tool to implement the filtering rules shown in Table 1, where its inputs are the RSF relations and the traces for selected features. Filters are applied to get Recipe Acitivity or Method Body Item. This tool applies different filters on the input to obtain the hot-spots and activities related to a feature instantiation about classes and methods, which follows the rules presented on Table 1 and the meta-model presented on the Figure 1.

The filtering process is in general performed according to the relationship between the example application classes (called SPECIFIC in RSF) with the hot-spots that are FRAMEWORK elements in RSF used by the application (in a SPECIFIC class or method), guided by the

[b] `https://github.com/arend-von-reinersdorff/java2rsf`
[c] `http://www.rigi.cs.uvic.ca/downloads/rigi/doc/node52.html`

Table 1: Rules and filters for instantiation activities. Legend: X, Y and W are classes/interfaces.

Rule	Filter Condition
C1) INTERFACE IMPLEMENTATION: X Y	Implements X Y and Specific X and Interface Y Framework Y and Trace X.anyMethod
C2) DIRECT SUBCLASSING: X Y	Extends X Y and Specific X and Framework Y and Trace X.anyMethod
C3) CLASS COLLABORATION: X Y	Specific X and Framework Y and (Method Invocation X.methA(parA) Y.methB(parB) or Class Instantiation X:methA(parA) Y.new) and not (Extends X Y or Implements X Y or Indirect Subclassing X Y)
C4) INDIRECT SUBCLASSING: X Y	Specific X and Framework Y and Extends X W_1 and ... and Extends W_n Y ($n \geq 1$) and Trace X.anyMethod
D1) METHOD INVOCATION: X.methA(parA) Y.methB(parB)	Calls X.methA(parA) Y.methB(parB) and Specific X and Framework Y and Method OF X methA(parA) AND Method OF Y methB(parB) AND Trace X.methA(parA) (n-1 stack level) AND Trace X.methB(parB) (n stack level)
D2) METHOD OVERRIDING: X.meth(par) Y.meth(par)	(Direct Subclassing X Y or Indirect Subclassing X Y) and Method OF X meth(par) and Method OF Y meth(par) and Trace X.meth(par)
D3) CLASS INSTANTIATION: X.meth(par) Y	Calls X.methA(par) Y.new and Specific X and Framework Y and Trace X.methA(par) (n-1 stack level) and Trace Y.new (n stack level)

recipe activities defined in the meta-model and filtered by the rules summarized in Table 1. Hot-spots candidates are obtained with the static analysis and they are confirmed with class and/or methods occurrence in the execution trace for the feature. A rule has been defined for each activity that composes the recipe.

Once the activities are obtained, the CookFrame organizes the information about all hot-spot classes and methods that could be used to instantiate the features into tables which provide the information for the recipes. In addition, we have information regarding to the classes and methods from example applications that can be used as source code examples to be part of the recipes.

3.2.4. ***Create recipes***

Documentation items regarding to the classes and methods from example applications using the hos-spots can be used. Those items are: E1) class declaration; E2) code comments; E3) design patterns; and, E4) code examples, as presented on the meta-model.

From the interests filtered in the previous step, which informs the hot-spots of interest to compose the recipes, it´s possible obtain those document itens that will compose the Cookbook. Those are extracted from source code or from archived information such

as design patterns results and RSFs. Document items are included on the recipes tables and the CookFrame tool converts those tables automatically into structured HTML files which composes the recipes. This step is represented in Figure 2 - E, and in the last step of the Algorithm 2: *Generate the Cookbook in the HTML format*. Cookbooks presents an index of recipes to implement features that can be realized with the framework. This index lists features and their description. Recipes are structured by features, considering that the developer usually starts the development from the requirements, which could be a list of features that must be implemented. When developers choose a feature to implement, they need to know which hot-spots to use and how, which means understanding their interactions within the framework. To address this problem, recipes present an index of hot-spots that could be used to implement a given feature and present activity to actually implement these hot-spots, based on existing application examples. This index shows the description of each hot-spot. This information is obtained from the hot-spot code comments. Cookbooks for the frameworks JHotDraw and JFace, tools for download and examples of RSF, traces and filtering scripts are available in a repository on GitHub [d].

4. Experimental design

To evaluate the extent of the approach contribution for the construction of framework documentation, we have conducted an experimental study, which compares the use of manually generated traditional documentation with the semi-automatically constructed cookbook, in terms of time and correctness to execute instantiation activity, and also in terms of user satisfaction. We decided to use traditional documentation as baseline instead of another framework documentation approach because a human-written baseline could be considered an upper-bound for any proposed automated approach, at least in the current state-of-the-art.

This section describes three experiments involving real instantiation activities. The unit of study in these experiments is the individual during the execution of framework instantiation activities, then it is possible compare the same individual supported by either the cookbook or the traditional framework documentation. Since, the subject performs two activities with the same framework, we mitigated the learning bias by choosing activities that rely on different parts (classes) of the framework. The detailed are described in the next sections. Qualitative and quantitative data were collected for subsequent analyses. The experiments were developed to answer the following research questions:

Q1: Does the use of the cookbook improve the time of instantiation activities when compared to the use of traditional framework documentation?

Q2: Are the solutions (code) produced for an instantiation activity less correct, more correct, or similar, when comparing the use of cookbooks and traditional documentation?

Q3: Was the information provided in the cookbook more useful, less useful, or similar to the usefulness of information in traditional documentation?

Q4: Is there some difference between the perception of activity complexity when comparing the use of cookbooks and traditional documentation?

[d] `https://github.com/lascam-UFU/CookFrame`

Q5: Is there some difference between the perception of usefulness of navigation structure in documents when comparing the use of cookbooks and traditional documentation?

Q6: Is there some difference between the degree of satisfaction attributed to the use of cookbooks and traditional documentation?

Q7: Is there some difference between the ease of localization of instantiation interests by comparing the two documentations: cookbook and traditional?

Q8: Is there some difference between the clarity of information comparing the two documentations: cookbook and traditional?

The first two research questions (Q1 and Q2) address quantitative objectives and were addressed in this way.

4.1. *Target frameworks*

Two Java frameworks were selected for this study: **JHotDraw 5.3** [e] (211 classes, 1924 methods and 11 packages), a framework for drawing diagrams systems and **JFace 3.4.2** [f] (271 classes, 4580 methods and 17 packages), a framework for creating interface views. Our choice for these frameworks is motivated by the following factors: these frameworks are used in related work; present good documentation to be used as baseline; it is possible to find example systems that use these frameworks; the source code of the framework is made available to enable static and dynamic analyses; and, they address application domains that will be understandable for most potential subjects.

The traditional documentation of JHotDraw is a structured HTML document with: *i)* a simplified class diagram with the 8 main framework classes; *ii)* a list of features that can be instantiated with the framework; *iii)* a package organization with reference to Javadoc; *iv)* an overview of example systems delivered by JHotDraw to aid in instantiation, available in framework code for users; *v)* instructions on how to compile the framework; *vi)* information on framework versions; *vii)* Javadoc with all packages, classes and methods of the framework.

JFace and SWT traditional documentation are available in the Eclipse Plugin Developer's Guide[g], which has a specific topic for the features to be developed in experimental activities. This guide presents: *i)* an introduction to building views using JFace, with possible features that can be implemented; *ii)* lists with hot-spot classes and explanation on their use to develop the views, including those required in experimental activities; *iii)* the class names linked to Javadoc with information about these classes, packages, their methods, and parameters; *iv)* several code examples throughout the document for view development, including useful examples for the experimental activities.

We performed experiments with isolated systems. In cases of multi-module systems running at the same time, the approach allows to isolate the interest based on the definition of which code are part of the framework (FRAMEWORK) or a system (SPECIFIC) part. However, when you have multiple systems running together, trace collection can collect

[e] `http://www.jhotdraw.org`
[f] `https://wiki.eclipse.org/JFace`
[g] `http://help.eclipse.org`

more false positives (FP) and a large amount of information. With that in mind, we recommend to work with a limited set of features, and divide the whole documentation into multiple feature groups. These false positives can be eliminated in the filter process, but requires more data to be analyzed. Another point is that larger systems use multiple languages, and we implemented the approach only for systems and frameworks implemented using the JAVA language. However, a similar approach could be developed for other languages.

4.2. *Subjects*

The subjects in this experiment are 14 Computer Science and Information Systems undergraduate students from the Institution 1; 15 Information Systems undergraduate students from the Institution 2; and 15 professionals, being 9 from academia and 6 from industry. All subjects are volunteers not previously involved with the research. The participation of graduate people from industry and academia is motivated by the concern of Di Penta et al. [24] that "a group composed of only students may not adequately represent the population of possible users".

We considered five important questions for the selection of subjects: i) level of Java knowledge; ii) knowledge regarding Eclipse IDE iii) knowledge concerning the use of frameworks, the subjects had not developed any application with the selected frameworks; iv) number of systems already developed by the students; v) and, professional experience. The students of Institution 1 have more experience than the students of Institution 2, both characterized mostly as subjects with little experience. The profile of the professionals who participated is very varied, presenting different levels of knowledge and experiences. The time of professional experience varied, with 33.33% of subjects having 4 to 6 years of experience, 40% subjects ranging from 10 to 13 years of experience, and 73,33% of subjects worked or currently work as Java developers.

Leveling was not performed among the subjects to compose the groups, since each subject performed two activities, one using the traditional framework documentation, and the other using the cookbook, according to Table 2 for the same experiment, mitigating the threat caused by different levels of knowledge and subject interest. One subjects part of first used the cookbook and the other party used the traditional approach first.

4.3. *Instantiation activities*

Two real framework instantiation activities were defined for each experiment, which could be supported by both documents (cookbook and traditional). One important threat to be mitigated in paired activities is related to subject alternation between the use of cookbook and traditional document. The first contact (activity) with the framework may influence the second contact. Therefore, we have defined pairs of activities that explored different and independent hot-spots. The activities were tested by a developer who did not participate in the experiments. In these tests, several activities were considered and measured in terms of time to be completed, in order to find out activities considered similar in time of execution and complexity. The complexity was measured considering developer perception,

Table 2: Organization of the experiments, subjects, activities, frameworks and documentation.

Experiment	Subject groups	#Subject	Activity	Framework	Document
Institution 1	A	7	Act. 1.1	JFace	Traditional
Institution 1	B	7	Act. 1.1	JFace	Cookbook
Institution 1	A	7	Act. 1.2	JFace	Cookbook
Institution 1	B	7	Act. 1.2	JFace	Traditional
Institution 2	C	8	Act. 2.1	JHotDraw	Traditional
Institution 2	D	7	Act. 2.1	JHotDraw	Cookbook
Institution 2	C	8	Act. 2.2	JHotDraw	Cookbook
Institution 2	D	7	Act. 2.2	JHotDraw	Traditional
Professional	E	8	Act. 3.1	JHotDraw	Traditional
Professional	F	7	Act. 3.1	JHotDraw	Cookbook
Professional	E	8	Act. 3.2	JHotDraw	Cookbook
Professional	F	7	Act. 3.2	JHotDraw	Traditional

and quantity of actions and hot-spots required to perform the activity. The complexity of the activities varied according to the experiment, since two of them were carried out with students and one with professionals with more experience. Moreover, the experiments with students were carried out in class schedules and this limited the complexity of the activities in order to make feasible the execution of the experiment within the time limit.

One more action was taken to minimize the impact of needing two activities to compare the two documentation, the two experiment activities were performed using the cookbook and using traditional documentation, as can be seen in the Table 2. For example, for the experiment with Institution 1, half of the subjects (group B) performed Activity 1.1 using the cookbook and the other half (group A) performed Activity 1.2 using the cookbook, according to Table 2. In this way, we have that the cookbook and the traditional documentation were evaluated by the same subjects performing the same group of activities. However, no subject performed the same activity twice. One part of the subjects used the cookbook first and the other part used the traditional approach first.

We define different types of activities. The activities performed per experiment are presented below:

Students Institution 1 - JFace: For this experiment, an incomplete Java class was given to subjects for each activity. They were instructed to complete this class using JFace framework to perform the activities: **Activity 1.1:** Create a simple table; **Activity 1.2:** Create a simple tree.

Students Institution 2 - JHotDraw: For this experiment, the subjects should make improvements to the JModeller system, with the inclusion of new functionalities. JModeller is a system that draws class diagrams and uses the JHotdraw framework. **Activity 2.1:** Create a menu item to change the color of the Class figure; **Activity 2.2:** Create a menu item to move the Class figure.

Professionals - JHotDraw : For this experiment, the subjects should make improvements in JModeller system, with the inclusion of new functionalities. **Activity 3.1:** Create

a new figure shape - hexagon; **Activity 3.2:** Create a new figure connector with a pentagon at the end.

4.4. *Environment preparation*

Real instantiation activities with frameworks require IDEs and specific platforms installed. We installed and tested these platforms in laboratories of the institutions before the experimental execution with the student subjects. For the experiment with the professionals, we sent instructions and systems by mail and subjects prepared and tested the environment before experimental execution. The minimum machine configuration was not defined for the experiments, as the software and Eclipse IDE typically run adequately on any developer machine or laboratory machine.

4.5. *Subject training*

Before executing activities, all subjects were instructed on experimental procedures. Documents regarding these procedures were created and sent to the subjects. For the professional subjects, these documents were more detailed as they performed the experiments remotely. After the initial instructions, the students were introduced to the structures of cookbook and traditional documentation for a best use of both. The training followed a structured script in order to present the same quality for all groups. Two videos were created to explain the documents and were sent to professionals with instructions. In these training sessions, subjects were not presented to the document contents related to experiment activities, and they had an average duration of 30 minutes, approximately 15 minutes for each documentation. The subjects were also instructed on the scope of the activities.

4.6. *Variables*

All experiments are bounded by the same dependent and independent variables. The *independent variable* in our experiments is the type of used documentation (cookbook or traditional) during the activities. Follow the dependent variables for each research question:

Q1: the dependent variable is the time spent to perform the maintenance activity.

Q2: the dependent variable is the correctness of the given activity. We analyzed the code resulted from activities for grading correctness. Using predefined test cases and source code analysis, grades were assigned within a score from 0 to 4, where: 0: could not accomplish anything of the activity; 1: concluded part of the activity, but not up to execution; 2: concluded the activity conceptually, but minor errors prohibited execution; 3: concluded the activity, but minor test cases failed; and, 4: concluded the activity passing all test cases.

Q3: the dependent variable is the perception of the subject concerning information usefulness provided by the document (cookbook or traditional).

Q4: the dependent variable is the perception of the subject concerning difficulty in activity execution.

Q5: the dependent variable is perception of the subject concerning usefulness of navigation resource on documents during the activities.

Q6: the dependent variable is perception of the subject concerning satisfaction using the documents during the activities.

Q7: the dependent variable is perception of the subject concerning facility in locating the instantiation interests in the documents during the activities.

Q8: the dependent variable is perception of the subject concerning information clarity in the documents.

The research questions **Q3-Q8** were answered through the activity questionnaire. The eventual differences on questions between the two groups was regarded to the usefulness of specific types of information presented in the documents. These questions were answered using the following *Likert* scale: 2 = Strongly Agree; 1 = Agree; 0 = Neutral - Did not know/opine; -1 = Disagree; -2 = Strongly disagree. For each closed survey question, there was an optional essay question getting the subject opinion aimed at better understanding the results. The questionnaire templates are available in the previously cited GitHub repository.

4.7. *Experimental execution*

The two experiments with student subjects followed the same procedures and were executed in the classroom monitored by the first author. The experiment with professional subjects, although following a similar procedure, was executed remotely. This flexibility aided in participation of professional subjects.

For the students, different schedules were defined for the experimental session, so they could more easily attend. The professionals executed the activities on a convenient day for them, but constrained by a predefined deadline for submitting the result.

For each experiment, subjects were separated into two groups. Each group was alternately instructed to use one documentation, according to Table 2. Subjects were trained. After the training, subjects were instructed to perform the first activity. Students had a time limit of 50 minutes. Professionals were instructed to execute the first activity on a day that they could devote adequate time to it, at least tree hours.

Professionals were informed that the time to perform the activity was unlimited. All subjects were instructed to collect the starting and ending time, and also possible pauses of activity execution. Time of training and instruction were not considered. They were also instructed to use the documentation and observe its usefulness during the activity; save the changed code for analysis; and, answer the activity questionnaire after finish the activity or timing out the execution. After completing the first questionnaire, students were introduced to the second activity that followed the same procedure. Professionals received the second activity, which followed the same procedure, after submitting the questionnaire and code for the first activity. The experimenter advised the participants that they could to give up at any time. The questionnaires over the activities were applied after activity execution.

4.8. *Analysis*

The Wilcoxon Signed Rank test was used in order to analyze data for questions Q1 to Q8. Wilcoxon Signed Rank test considers paired data, i.e., the test compares the values of the same subject in different groups. We also use Cliff's δ to measure the effect size, whenever

Table 3: Results of experiments based on Wilcoxon Signed-Rank test and Cliff's δ effect size.

Criterion	**Students Inst. 1**		**Students Inst. 2**		**Professionals**	
	p-value	Cliff's δ	p-value	Cliff's δ	p-value	Cliff's δ
(Q1) Time spent	**0.036**	0.32 (small)	1	*	0.5894	*
(Q2) Correctness	**0.002**	-0.82 (large)	0.371	*	0.5862	*
(Q3) Information Usefulness	0.057	-0.48 (large)	**0.019**	-0.46 (medium)	0.3929	*
(Q4) Complexity of Activity	0.067	0.39 (medium)	0.590	*	0.548	*
(Q5) Usefulness of Navigation	**0.033**	-0.48 (large)	0.070	-0.38 (medium)	**0.039**	-0.48 (large)
(Q6) Documentation Satisfaction	0.104	*	**0.030**	-0.50 (large)	**0.021**	-0.43 (medium)
(Q7) Facility to Locate Information	0.131	*	**0.042**	-0.38 (medium)	**0.043**	-0.52 (large)
(Q8) Clarity of Information	0.943	*	**0.025**	-0.40 (medium)	0.205	*

we have significant or almost significant differences at p-value of 0.05. Negative values for δ indicate that the cookbook group has higher values for the tested measure.

5. Results

In this section, we present the experimental results for the respective research questions relying on the statistical tests shown in Table 3.

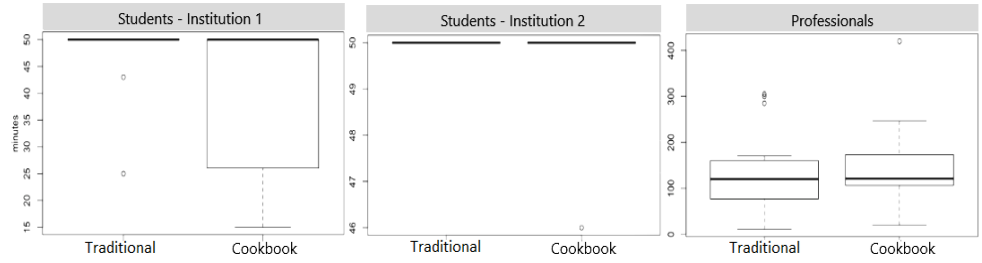

Fig. 3: Boxplots of time spent executing activities (Q1).

Q1: About the **time spent**, at experiment with students from institution 1, the results showed a significant difference with best results for the cookbook. More importantly, six out of 14 using cookbooks could finish before 50 minutes, whereas only two out of 14 using

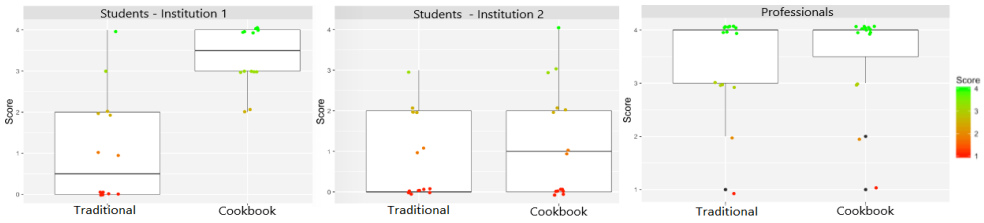

Fig. 4: Boxplots for activities' correctness (Q2).

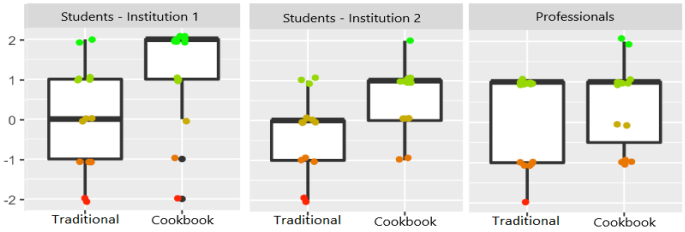

Fig. 5: Boxplots of information usefulness (Q3).

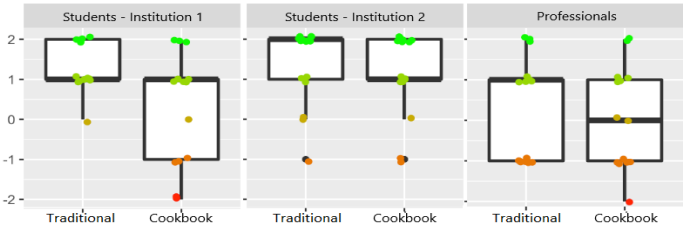

Fig. 6: Boxplots of perception of complexity (Q4).

traditional documentation could do that. At experiments with students from institution 2 and Professionals, the tests presented no significant difference for the time spent executing activities as can be seen in Table 3 and Figure 3.

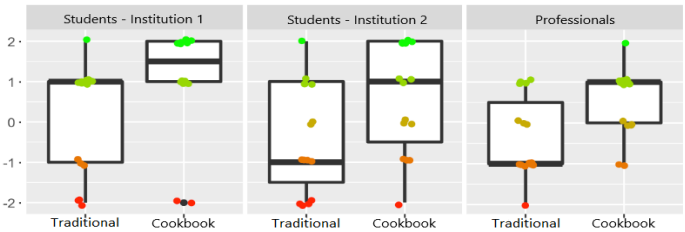

Fig. 7: Boxplots of perception of usefulness of navigation (Q5).

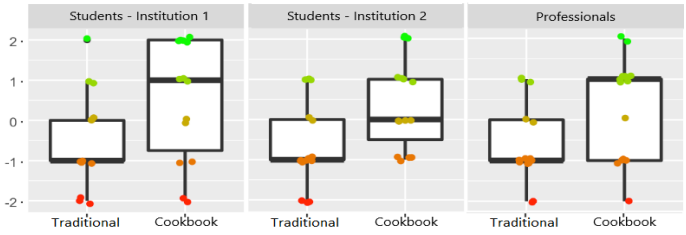

Fig. 8: Boxplots of documentation satisfaction (Q6).

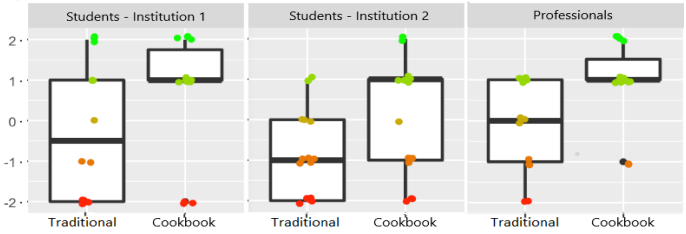

Fig. 9: Boxplots of facility to locate information (Q7).

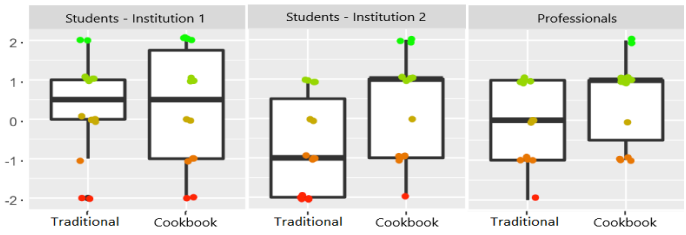

Fig. 10: Boxplots of clarity of information (Q8).

Q2: Students from institution1 showed a significant higher correctness rate, with large effect size, within the cookgroup group (Table 3). Students from institution 2 and professionals presented no significant difference in the correctness using cookbooks or traditional documentation, although the boxplots (Figure 4) are slightly favourable for the cookbook group.

Q3: The cookbook group presented a greater **perception of usefulness** of the documents information with a significant difference with medium effect for institution 2, and significant difference at p-value 0.057 with large effect for institution 1. The perception of information usefulness showed no significant difference among professionals.

Q4: The perception of complexity of the activities presented values with no significant difference with students from institution 2 and professionals. For students from institution 1, the test show a **lower** perception of **activity complexity** using the cookbook with

medium effect size when compared to traditional documentation, significant at p-value of 0.067. Note the in this case the effect size is positive, and it is favourable to cookbooks because the lower perception of complexity, the better.

Q5: The **usefulness of navigation** mechanism in the documents showed a significant difference at experiments with institution 1 and professionals with large effect size. For students from institution 2, cookbooks are also perceived to have useful navigation (Figure 7), significant at p-value 0.07, with medium effect size.

Q6: The **satisfaction with the use of the documents** showed a significant difference with students from institution 2 (large effect) and professionals (medium effect), between the results from each document group. There was no significant difference with students from institution 1, although data in Figure 8 show higher median for the cookbook group.

Q7: The **facility to locate information** was a significantly better within the cookbook group with students from institution 2 (medium effect) and professionals (large effect). There was showed no significant difference with students from institution 1 although Figure 9 shows higher median for the cookbook group.

Q8: The **clarity of information** was perceived as higher in the cookbook group with students from institution 2 with a medium effect. For students from institution 1 and professionals there was no perceived significant difference.

6. Discussion

The time spent to execute the activities showed a significant difference in the experiment with students of institution 1 (JFace), in favor of cookbooks. The activities' correctness was also significantly better for the cookbook group in this experiment. For this experiment, only one incomplete class was passed to each activity, and both documents presented code examples that could be used to help the activity, suggesting that code examples would be more easily accessible in the cookbook. For the other experiments, the activity correctness presented no significant difference using the different documentation. For the experiment with the professionals (JHotDraw), the time was not delimited, and there was no significant difference between the groups. Most of the professionals using the cookbook and traditional documentation successfully completed the activity. But, the essay reports indicate that they used the documents to locate the code samples and then preferred to use source code samples present on the target system in both activities, that is probably why the result were so similar using both documents. The source code examples were available in the application that should be changed, for the experiments with professionals and institution 2.

In the experiment with students from Institution 2 (JHotDraw), the instantiation activities were considered complex by subjects using both documents, due to students' level of experience.

Regarding information usefulness, cookbooks were significantly more useful in experiment with students from institutions 1 and 2. For the institution 1, the documents were the only support to execute the activities. The subjects from institution 2 were less experienced than professional, and this could have lead them to depend more on documentation to solve the activities. The proposed cookbook is structured by features that can be implemented,

presenting hot-spots with code examples. This structure supported by code examples of use is a differential characteristic from traditional documentation that seems to have influenced subjects perception of greater information usefulness for cookbooks compared to the traditional document.

Regarding activity complexity perceived, subjects using the cookbook had a perception of significantly lower activity complexity for experiments within institution 1 (JFace). This suggests that the cookbook may have assisted them in activities execution, leading them to a higher correctness, greater perception of information usefulness and consequently smaller perception of activity complexity. In the other experiments, there was no significant difference between the use of both documents. In the experiment with students from institution 2, most of the subjects evaluated the activities as complex using the both documents. The subjects presented difficulties to perform a simple activity of framework instantiation due to their inexperience. In the experiment with professionals, most of the subjects used the source code examples available in the target system in both activities. This may explain why there was no significant difference in perception of activities' complexity using both documents.

The information localization facility was significantly greater for the cookbook in experiments with students from institution 2 and professionals, which used the JHotDraw framework and the same document. The experiment with students from institution 1 (JFace) did not present a significantly different result. Probably the information localization facility is influenced by documentation structure and by the presence of necessary information. The location of the information in the cookbook is given by its structure, which is organized by recipes for features that can be instantiated, with the respective hot-spots that can be used. In these traditional documentation of JHotDraw, the hot-spots are presented by packages and in alphabetic order, and this makes it difficult to associate them with desired features. To help in this regard, the traditional document drives the subjects to source code examples available within the framework. The fact that the cookbook does not present a significant difference to information localization facility compared to the traditional JFace documentation is satisfactory, since JFace has a very well-structured documentation, where the important features to activities were presented separately by topics presenting hot-spots and code examples, with artifacts of navigation and search.

For all experiments, the cookbook obtained assessment of usefulness of navigation significantly better than the traditional documentation. The cookbook structure, where recipes organized features of interest matches the semantics of the activity, and thus is likely to facilitate the navigation in the document. The subject's answers indicated this. Following we highlight some answers: *"The document (cookbook) looked cleaner, bringing only what was really needed"*; the cookbook is *"divided by possible activities, It was necessary only to seek an activity (sic, recipe) that had the similar concept of proposed activity"*.

The information clarity presented a better result for the cookbook group in the experiment with students at institution 2 (JHotDraw) , and in the other experiments did not present a significant different result between the use of both documents. This can be considered satisfactory for a semi-automatically generated approach. This variable did not follow a trend and did not receive comments from the subjects that helped to evaluate the possible reasons

for this result.

Concerning satisfaction of documentation usage, in the experiments with students from institution 2 and professionals (JHotDraw), the results show significantly greater satisfaction using the cookbook compared to the traditional document. In the experiment with institution 1, there was no significant difference. This is similar as the information localization results, suggesting that a more subjective concept of satisfaction is coherent with the facility of localization.

Code examples usefulness was confirmed by the results and subjects' answers, who commented on the importance of examples for executing activities. In addition, the professional subjects indicate a preference for using code examples rather than documentation during the execution of instantiation activities, when they easily located these examples on the source code. Also, it is important to note that both documents helped to locate hot-spots and code examples for the professionals. See some subject reports: *"it (the cookbook) was helpful to know how to instantiate the Tool object"*; *"the examples facilitate the understanding of the hot-spots"*; *"through the examples (in documentation) it was possible to go directly to the class that could be used to do the activity"*.

6.1. *Threats to validity*

Internal validity - **subjects:** *i)* First, the subject performance could be influenced by their willingness at the moment. To mitigate this threat, we defined different days and schedules for student participation. Professionals subjects had a period longer than 20 days to execute each experiment. They could choose the place, day and time better suited to execute activities. In addition, all subjects were volunteers. *ii)* The subjects familiarity with documents and development environment may interfere with the results. A brief training about documents and contact with the used development environment was applied to subjects. But, learning and adaptation can vary for subjects. To mitigate that, all subjects participated using both documentations and results were paired. *iii)* Questions related to utility, complexity and importance present a qualitative nature related to perception that each subject has about these definitions. This threat is also mitigated by pairing the data. *iv)* Subjects may have some knowledge about the experimental goal. To mitigate this threat impact, no subjects involved with the research participated in experiments. The subjects were not familiar with research goals.

Internal validity - **miscellaneous:** *i)* The documents contents and sizes may influence the ease to navigate and locate information influencing the time spent and correctness. We mitigated this threat either creating cookbook with size close to traditional document, or, we limited the size of traditional documentation considering the part with information relevant to activities. *ii)* Experimenter effects may occur, since the experimenter is involved with the proposal and is interested in its success. All experimental procedures were planned and executed seeking equality between experimental groups.

External validity: *i)* The generalization of our results can be hampered by the limited representativeness of the subjects and the activity. The sample is varied with students, academic and software industry professionals. However, most of them are students, because,

it is difficult to get participation of industry professionals [3]. *ii)* Activities do not cover all possibilities of framework instantiations. Covering all possibilities would require the definition of many activities. Nonetheless, relevant instantiation activities were covered.

Construction validity: *i)* The first contact with the framework can influence the second contact. We mitigated that threat defining activities that implied in exploration of different hot-spots and different code parts. *ii)* The system to be modified presents code examples using JHotDraw were made available examples of features that could help the developer during activities. Subjects were advised to use and evaluate the documents. Nonetheless, this decision helped us to evaluate if the documents would be useful even when useful code examples are available.

7. Conclusion

This work has proposed and assessed an approach to generate documentation for framework instantiation based on a central idea of reverse engineering systems that use the corresponding framework. The documentation was organized using a cookbook format, whose recipes are differential because they organize activities, information, design patterns and examples of instantiation in a single documentation driven by features of interest.

We conducted three experiments with 44 human subjects executing 88 real framework instantiation activities to compare the generated cookbooks with traditional documentation. The time spent on activities using both documents did not present significant difference between the use of both documents for any experiment. The activities correctness was significantly better for cookbook group in experiment with students from Institution 1. Others experiments presented activities correctness result without significant difference between the use of both documents. The perception of satisfaction in document use was higher for the cookbook group in all experiments, which may have been influenced by the perception of information usefulness, easiness of location of information, navigation mechanism usefulness, and clarity of information that were better or with non-significant difference when compared to the traditional documents. The subjects' essay answers indicated the importance of code examples to carry out the instantiation activities. Professional subjects had a preference for using the code examples presented in sample systems rather than using the documentation. Nonetheless, for the Professionals, the results of satisfaction with documentation use and essay answers indicated that the documents were useful in locating hot-spots and sample code.

In the end, the proposed approach generates a documentation that has been considered to have a more useful navigation overall and a more useful content for students. Moreover, in the other aspects, cookbooks are, at least, similar compared to traditional documentation. So, this approach can be effective, either for situations where the documentation is missing or this is outdated, or during manual documentation writing using the approach output as a starting point.

As future work, the approach can be applied to other popular frameworks, such as, Android. One challenge in that case is the selection of adequate applications to serve as didactic examples to be part of the cookbook. Another possibility of future work is to

incorporate information from other resources, such as, Q&A sites, given they have already been shown to be viable [30, 27, 6].

Acknowledgment. We acknowledge Brazilian research agencies CAPES, CNPq, and FAPEMIG for partially funding this research. We would like to thank the 44 subjects for their participation.

References

[1] James D. Arthur and K. Todd Stevens. Document Quality Indicators: A Framework for Assessing Documentation Adequacy. *Journal of Software Maintenance: Research and Practice*, 4(3):129–142, 1992. ISSN 1096-908X.

[2] M. Bruch, T. Schäfer, and M. Mezini. Fruit: IDE support for framework understanding. In *Proc. of OOPSLA'2006*, pages 55–59, New York, NY, USA, 2006. ACM.

[3] B. Cornelissen et al. A controlled experiment for program comprehension through trace visualization. *IEEE TSE*, 37(3):341–355, May 2011.

[4] R. Cottrell et al. Compare and contrast: Visual exploration of source code examples. In *Proc. of the 5th IEEE VISSOFT'2009.*, pages 29–32, Sept 2009.

[5] B. Dagenais and H. Ossher. Automatically locating framework extension examples. In *Proc. of SIGSOFT'2016*, pages 203–213, New York, NY, USA, 2008. ACM.

[6] F. Delfim et al. Redocumenting APIs with crowd knowledge: a coverage analysis based on question types. *Journal of the Brazilian Computer Society*, 22(1):9, 2016.

[7] T. Eisenbarth, R. Koschke, and D. Simon. Locating features in source code. *IEEE Transactions on Software Engineering*, 29(3):210–224, 2003.

[8] B. Cornelissen et al. A systematic survey of program comprehension through dynamic analysis. *IEEE TSE*, 35(5):684–702, Sept 2009.

[9] M. Fayad and D. Schmidt. Object-oriented application frameworks. *Commun. ACM*, 40(10):32–38, October 1997. ISSN 0001-0782.

[10] S. Henninger. Retrieving software objects in an example-based programming environment. In *Proc. of SIGIR'1991*, pages 251–260, New York, NY, USA, 1991. ACM.

[11] A. Heydarnoori et al. Two studies of framework-usage templates extracted from dynamic traces. *IEEE TSE*, 38(6):1464–1487, Nov 2012.

[12] R. Holmes and G. Murphy. Using structural context to recommend source code examples. In *Proc. of ICSE'2005.*, pages 117–125, May 2005.

[13] J. Jiang et al. Constructing usage scenarios for API redocumentation. In *Proc. of ICPC'2007.*, pages 259–264, June 2007.

[14] R. Johnson. Components, frameworks, patterns. In *Proc. of SSR'1997*, pages 10–17, New York, NY, USA, 1997. ACM.

[15] Jinhan Kim, Sanghoon Lee, Seung-Won Hwang, and Sunghun Kim. Enriching Documents with Examples: A Corpus Mining Approach. *ACM Trans. Inf. Syst.*, 31(1): 1:1–1:27, January 2013. doi: 10.1145/2414782.2414783.

[16] D. Kirk, M. Roper, and M. Wood. Identifying and addressing problems in object-oriented framework reuse. *Empirical Soft. Eng.*, 12(3):243–274, June 2007.

[17] E. Krasner and T. Pope. A cookbook for using the model-view controller user inter-

face paradigm in smalltalk-80. *J. Object Oriented Program.*, 1(3):26–49, aug 1988.

[18] R. F. Q. Lafetá, M. A. Maia, and D. Röthlisberger. Framework instantiation using cookbooks constructed with static and dynamic analysis. In *2015 IEEE 23rd International Conference on Program Comprehension*, pages 125–128, May 2015. doi: 10.1109/ICPC.2015.21.

[19] M.. Maia and R. Lafetá. On the impact of trace-based feature location in the performance of software maintainers. *JSS, 2013*, 86(4):1023 – 1037, 2013.

[20] M. Markiewicz and C. de Lucena. Object oriented framework development. *Crossroads*, 7(4):3–9, jul 2001.

[21] A. Michail. Data mining library reuse patterns using generalized association rules. In *Proc. of ICSE'2000*, pages 167–176, 2000.

[22] L. R. Neal. A system for example-based programming. *SIGCHI Bull.*, 20(SI):63–68, March 1989. ISSN 0736-6906.

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

[24] M. Di Penta, R.Stirewalt, and E. Kraemer. Designing your next empirical study on program comprehension. In *Proc. of ICPC'2007*, pages 281–285, June 2007.

[25] Martin P. Robillard. *Representing concerns in source code*. PhD thesis, Department of Computer Science, The University of British Columbia, Vancouver, Canada, 2003.

[26] Martin P. Robillard. What Makes APIs Hard to Learn? Answers from Developers. *IEEE Software*, 26(6):27–34, November 2009. ISSN 0740-7459.

[27] A. Rocha and M. Maia. Automated API documentation with tutorials generated from Stack Overflow. In *Proc. of the 30th Brazilian Symp. on Soft. Eng.*, SBES '16, pages 33–42, New York, NY, USA, 2016. ACM.

[28] F. Shull, F. Lanubile, and V. Basili. Investigating reading techniques for object-oriented framework learning. *IEEE TSE*, 26(11):1101–1118, Nov 2000.

[29] Karl L. Smart. Assessing quality documents. *ACM Journal of Computer Documentation*, 26(3):130–140, August 2002. ISSN 1527-6805.

[30] L. Souza, E. Campos, and M. Maia. On the extraction of cookbooks for APIs from the crowd knowledge. In *Proc. of Brazilian Symp. on Software Eng.*, SBES'2014, page 10pp, Maceió, Brazil, 2014.

[31] Lucas B.L. Souza, Eduardo C. Campos, Fernanda Madeiral, Klérisson Paixão, Adriano M. Rocha, and Marcelo de Almeida Maia. Bootstrapping cookbooks for apis from crowd knowledge on stack overflow. *Information and Software Technology*, 111:37 – 49, 2019.

[32] C. Treude and M. Robillard. Augmenting api documentation with insights from stack overflow. In *Proc. of ICSE'2016*, pages 392–403, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-3900-1.

[33] C. Treude, M. Robillard, and B. Dagenais. Extracting development tasks to navigate software documentation. *IEEE TSE*, 41(6):565–581, June 2015. ISSN 0098-5589. doi: 10.1109/TSE.2014.2387172.

[34] N. Tsantalis et al. Design pattern detection using similarity scoring. *IEEE TSE*, 32

(11):896–909, Nov 2006.

[35] G. Uddin and M. P. Robillard. How api documentation fails. *IEEE Software*, 32(4): 68–75, July 2015.

[36] Y. Ye, G. Fischer, and B. Reeves. Integrating active information delivery and reuse repository systems. *Proc. of SIGSOFT'2000*, 25(6):60–68, November 2000. ISSN 0163-5948.

[37] Z. Zhu et al. Automatically generating task-oriented api learning guide. In *Proc. of Asia-Pacific Symp. on Internetware'2017*, pages 12:1–12:10, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5313-7.