

Patterns for Documenting Frameworks – Part II

Authors **Ademar Aguiar, Gabriel David**
FEUP & INESC Porto, Universidade do Porto
E-mail: ademar.aguiar@fe.up.pt, gtd@fe.up.pt

Good design and implementation are necessary but not sufficient pre-requisites for the successful reuse of object-oriented frameworks. Although not always recognized, good documentation is crucial for effective framework reuse and comes with many issues. Defining and writing good quality documentation for a framework is often hard, costly, and tiresome, especially when not aware of its key problems and the best ways to address them. This document presents patterns from a set of related patterns that describe proven solutions to recurrent problems of documenting object-oriented frameworks. The pattern language they all form together aims at helping non-experts on cost-effectively documenting object-oriented frameworks. The patterns here presented address the problems of *explaining how to use* a framework and *illustrating what it can be good for*, respectively the patterns “COOKBOOK & RECIPES” and “GRADED EXAMPLES”.

Introduction Object-oriented frameworks are a powerful technique for large-scale reuse capable of delivering high levels of design and code reuse. As software systems evolve in complexity, object-oriented frameworks are increasingly becoming more important in many kinds of applications, new domains, and different contexts: industry, academia, and single organizations.

Although frameworks promise higher development productivity, shorter time-to-market, and higher quality, these benefits are only gained over time and require up-front investments. Before being able to use a framework successfully, users usually need to spend a lot of effort on understanding its underlying architecture and design principles, and on learning how to customize it, which all together implies a steep learning curve that can be significantly reduced with good documentation and training material.

This paper contributes with two patterns to the work in progress of writing a pattern language that focus on problems of documenting frameworks [1], some of the several technical, organizational, and managerial issues that must be well managed in order to employ frameworks effectively.

Pattern language The pattern language comprises a set of interdependent patterns aiming at helping developers on becoming aware of the typical problems they will face when documenting object-oriented frameworks. The patterns were mined from existing literature, lessons learned, and expertise on documenting frameworks, based on a previous compilation about framework documentation [2].

The pattern language describes a path commonly followed when documenting a framework, not necessarily from start to end to achieve effective results. In fact, many frameworks are not documented as completely as suggested by the patterns, due to different kinds of usage (white-box or black-box) and different balancing of tradeoffs between cost, quality, detail, and complexity. One of the goals of these patterns is precisely to expose such tradeoffs in each pattern, and to provide practical guidelines on how to balance them to find the best combination of documents to the specific context at hands.

According to the nature of the problems addressed, the patterns are organized in *artefact patterns* (*which kinds of documents to produce? what should they include? how to relate them?*) to which belong the patterns here documented, and *process patterns* strictly related with the process of cost-effectively documenting frameworks (*how to do it? which activities, roles and tools are needed?*).

Artefact patterns Artifact patterns address problems related with the documentation itself, here seen as an autonomous and tangible product independent of the process used to create it. They provide guidance on choosing the kinds of documents to produce, how to relate them, and what to include there.

Similarly to other technical documentation, the overall quality of framework documentation is complex to determine and assess, and this is perhaps the first issue. Documentation must have quality, that is, it must be easy to find, easy to understand, and easy to use [3]. Task-orientation, organization, accuracy, and visual effectiveness are among all documentation quality attributes, the most difficult ones to achieve on framework documentation [2].

From the reader’s point of view, the most important issues are on providing accurate task-oriented information, well-organized, understandable, and easy to retrieve with search and query facilities. From the writer’s point of view, the key issues are on selecting the contents to include, on choosing the best representation for the contents, and on organizing the contents adequately, so that the documentation results of good quality, while easy to produce and maintain.

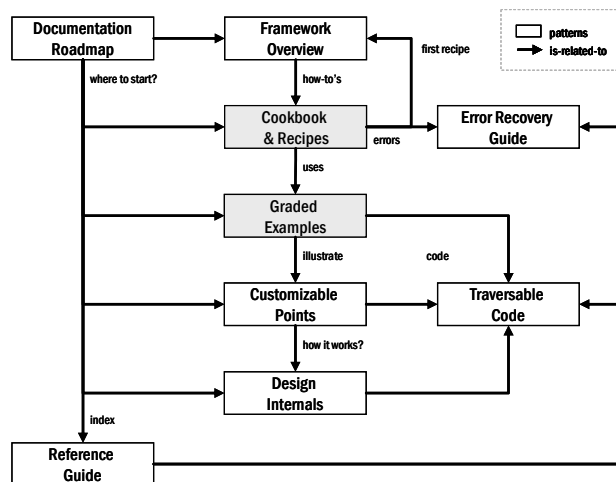


Figure 1. Documentation artefact patterns and their relationships.

Patterns overview To describe the patterns, we have adopted the Christopher Alexander's pattern form: *Name-Context-Problem-Solution-Example* [4]. Before going to the detail of each pattern, we will overview the pattern language with a brief summary of each pattern's intent. For contextual purposes, all the artifact patterns are overviewed below and depicted in Figure 1. highlighting the two patterns described in this paper.

Documentation Roadmap helps on deciding what to include in the documentation overview to provide readers of different audiences with useful and effective hints on what to read to acquire the knowledge they are looking for [1].

Framework Overview suggests providing introductory information, in the form of a framework overview, briefly describing the domain, the scope of the framework, and the flexibility offered, because contextual information about the framework is the first kind of information that a framework user looks [1].

Cookbook & Recipes explains how to provide readers with information that explain how-to-use the framework to solve specific problems of application development, and how to combine this prescriptive information with small amounts of descriptive information to help users on understanding what they are doing.

Graded Examples describes how to provide and organize example applications constructed with the framework and how to cross-reference them with the other kinds of artefacts (cookbooks, patterns, and source code).

Customizable Points describes how to provide readers with task-oriented information with more precision and more design detail than cookbooks and recipes, so that readers can quickly identify the customizable points of the framework (hot-spots) and to understand how they are supported (hooks).

Design Internals explains how to provide detailed information about what can be adapted and how the adaptation is supported, by referring the patterns that are used in its implementation and where they are instantiated.

Reference Guide suggests what to include as reference information and how to structure the documentation to make it the most complete and detailed as possible to assist advanced users when looking for descriptive information about the artefacts and constructs of the framework.

Traversable Code provides hints on how to organize and present source code, both of the examples and the framework itself, when desired, to make it easy to browse and navigate, from, and to, other software artefacts included in the overall documentation, namely models and documents.

Error Recovery Guide explains how to help users on understanding and solving the errors they encountered when using the framework.

Pattern **Cookbook & Recipes**

The quality of information explaining how-to-use the framework is very important to achieve its effective reuse. In most of the cases, the informal communication channels between framework developers and framework users are not available anymore, and, as a result, all information has to be communicated in alternative ways, with different kinds of manuals delivered with the framework, possibly using different media (documents, animations, videos).

Problem To help application developers being effective on the reuse of a framework, ideally, the documentation should be organized in a way that can help readers quickly locate the explanations they strictly need to learn how to use the framework parts required to implement the specific features of the application at hands.

How to help readers on quickly learning how to use a framework?

Forces **Task-orientation.** Readers want to learn how to use the framework, so the documentation must focus on real tasks that users really want to perform, instead of artificial tasks, imposed by the framework.

Balancing Prescriptive and Descriptive information. To be effective, the documentation must achieve a perfect balance between the level of detail of the instructions provided to guide the usage of the framework, and the level of detail and focus used to communicate how the framework works, i.e. its design internals. This perfect balance may vary with the reader of the documentation, and thus the “one size fits all” solution doesn’t work here.

Different Audiences. Readers of different audiences have different needs and interests that must be addressed by the documentation. New framework users want to identify, understand and manipulate the flexible features of the framework they need, as quick as possible, without being forced to understand the detail of the whole design, but only its basic architecture (static and dynamic views). More advanced users may want to learn how to do less common customizations that possibly require a more detailed understanding of the framework’s internal design.

Completeness. Readers appreciate complete information, i.e., that all possible customizations are mentioned, with all the possible detail, but this is not always feasible, as it largely depends on the reader’s point of view and the tasks to support.

Easy-to-use. The resulting documentation must be easy to use, otherwise readers would need to spend more time than needed to browse it, and can get lost on it.

Solution Provide a collection of *recipes*, one for each framework customization, organized in a *cookbook*, which acts as a guide to the contents of all its recipes.

Cookbook. A good cookbook is specific to a particular framework. Users search the cookbook for the recipe that is most appropriate for their needs, and when found, they follow the steps it describes. Recipes in a cookbook are usually organized in a spiral approach, where the most common forms of reuse are presented early, and

concepts and details are delayed as long as possible. A *framework overview* is often the first recipe in a cookbook [5], being responsible for presenting the framework.

Recipe. A recipe is a document that informally describes how to use the framework to solve a specific problem of application development [6]. Recipes present information in natural language, perhaps with some pictures and fragments of source code. Although recipes are rather informal documents, they are usually structured in:

- a purpose section
- a how to do section containing a sequence of steps to follow, and
- explicit, or implicit, references to other recipes, models, examples and fragments of source code.

The most important kind of information provided by cookbooks and recipes is prescriptive information, which instructs users on how to use and customize the flexibility features provided by the framework. In addition, they also informally describe architectural constructs and design details, but only in a very small amount, the amount strictly needed to help users understand minimally what they are doing.

The best kind of documentation for beginners seems to be one that provides detailed instructions for using each individual feature of the framework without describing in detail all the theory behind them. Considering that the main purpose of a framework is to reuse design, if it is well designed, then there must exist large parts of its design that are easy to reuse even without knowing them.

On the other hand, to start using a framework without having a clear understanding of it seems to be wrong at first, but the fact is that people can't understand well a framework until they have really used it. The effective understanding of a framework thus requires that theory follows practice [7]. After the first use, the framework user has a much better understanding of what the framework does and is more capable to understand how it works, i.e. to understand its internal design details.

Variants **Active cookbook.** The active cookbook is a specialization of the cookbook concept that provides active guidance to framework users [7]. Active cookbooks extend the cookbook idea with a hypertext representation, a visual development environment and supporting tools. The tool support provided by active cookbooks helps the user navigate the steps of the recipes and provide the tools needed in each step, thus increasing the productivity.

Johnson's patterns. Johnson documented the HotDraw framework [7] using a pattern language comprising a set of patterns, one pattern for each recurrent problem of using the framework. The pattern language organizes the documentation, as a cookbook does with the recipes, and each pattern provides a format for each recipe.

To avoid confusion with design patterns, the term motif was later introduced in [8] to name Johnson's patterns [7]. The description of a motif has sections similar to a recipe, except that use additional references to design patterns, to provide

information about the internal architecture, and references to contracts for a more rigorous description of the collaborations relevant to the motif.

Hook description. Hook descriptions were first introduced in [9] and present knowledge about framework usage, providing an alternative view to design documentation. Hook descriptions provide solutions to very well-defined problems. They detail how and where a design can be changed: what is required, the constraints to follow, and effects that the hook will impose, such as configuration constraints. A hook description usually consists of a name, the problem the hook is intended to solve, the type of adaptation used, the parts of the framework affected by the hook, other hooks required to use this hook, the participants in the hook, constraints, and comments. Hooks can be organized by hot spot: a hot spot tends to have several hooks within it. The usage of hooks can be semi-automated.

Examples Cookbooks and recipes were historically the first technique used to document frameworks, namely the MVC framework [5] and the MacApp framework [10].

JUnit. Figure 2. presents a recipe for writing a simple test with JUnit, the first of the five recipes contained in the cookbook provided with JUnit, the “JUnit Cookbook” document [11].

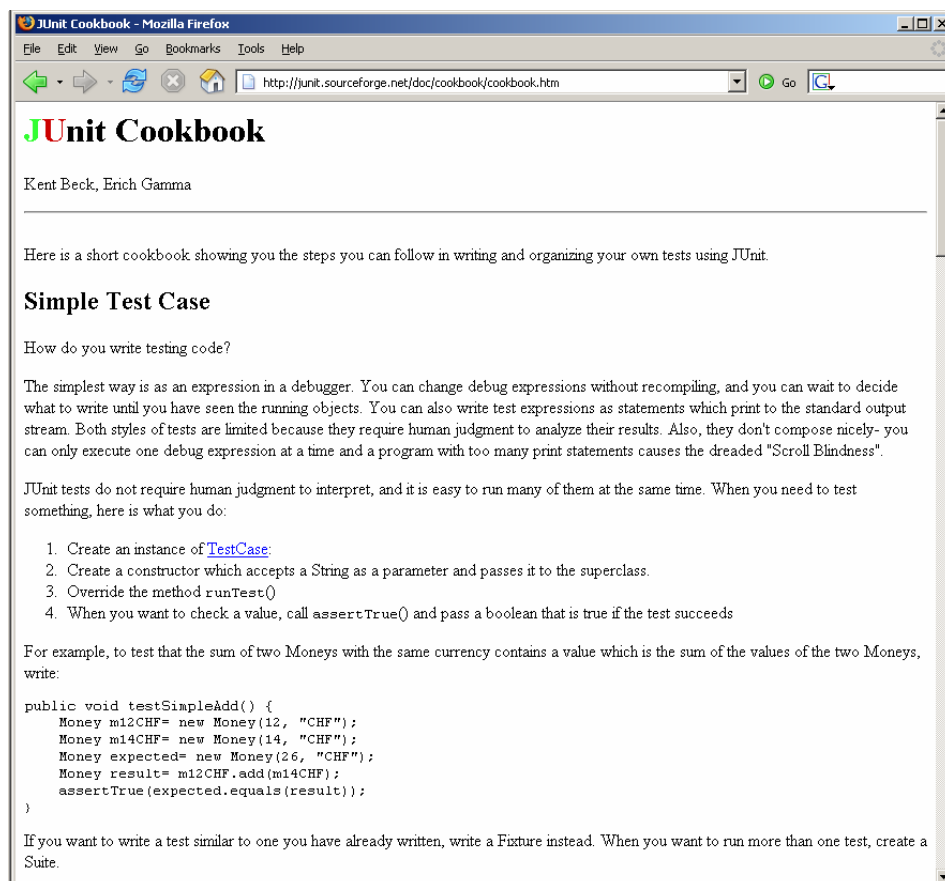


Figure 2. Example of a recipe for writing a simple test with JUnit.

Swing. The Sun's JFC/Swing framework [12] contains several recipes organized in several cookbooks. Figure 3. shows the cookbook related with layout managers, which contains a set of recipes explaining how to use layout managers. Figure 4. shows a specific recipe explaining how to use a BorderLayout manager.

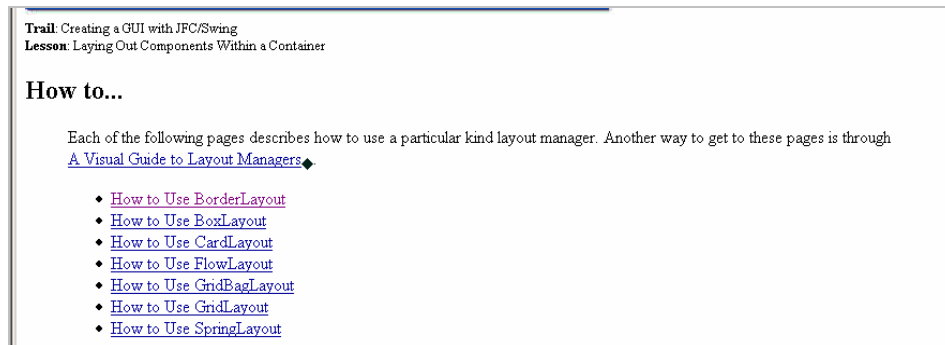


Figure 3. Example of a cookbook from Swing framework to teach how to use layout managers.

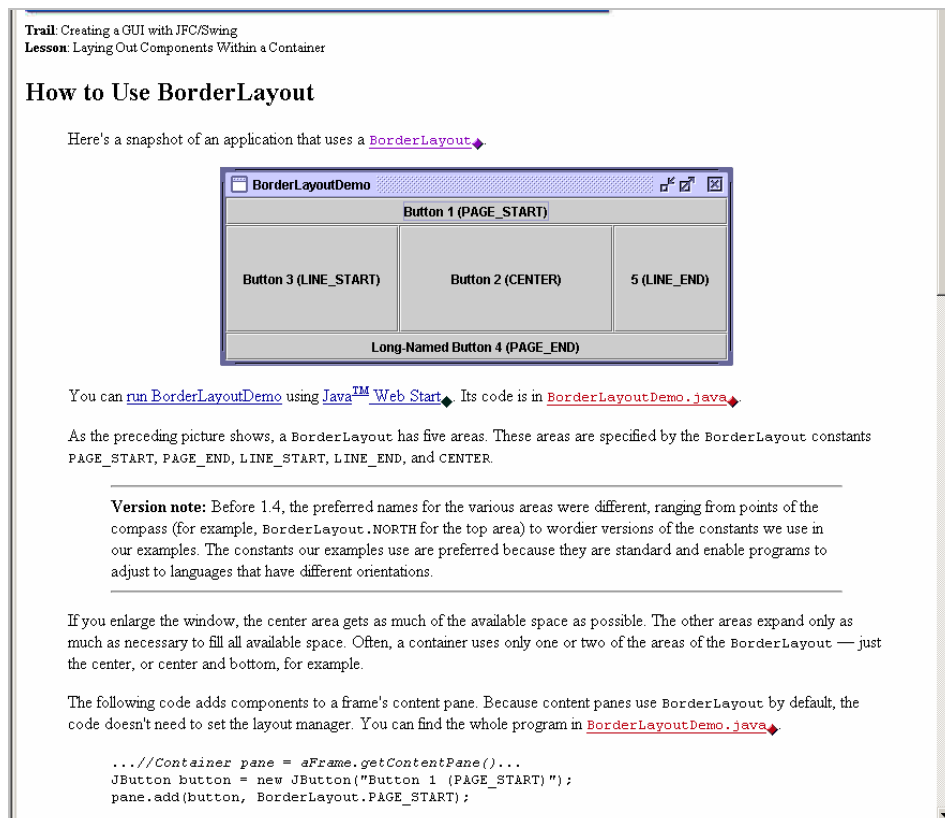


Figure 4. Example of a recipe from Swing framework to teach how to use the BorderLayout manager.

Consequences Providing a cookbook and recipes for a framework helps improve its task-orientation, usability, and helps on combining prescriptive with descriptive information.

Because cookbook and recipes focus on teaching how-to-use a framework, it is useful to combine them with concrete or running examples, and references to the related *customizable points* and *source code*.

Pattern **Graded Examples**

Information explaining what a framework can be used for is of great value for potential users, as it helps them to evaluate the appropriateness of the framework for the application at hands, and thus fundamentals the selection, or rejection of a framework.

Problem The first usage of a framework is always the hardest one as it may simultaneously involve the learning of the problem domain, the domain covered by the framework, and the range of solutions for which the framework was designed and is applicable. Subsequent usages of a framework can be easier

How to help readers on evaluating the appropriateness of a framework to his particular application at hands?

How to help readers on getting started fast using a framework?

Forces **Task-orientation.** Framework users want practical information illustrating what can be done with the framework, and how-to-use it.

Different Audiences. A framework selector is someone (manager, project leader, developer) who is responsible for deciding which frameworks to use in an application development project. Among other information, framework selectors will look for the domain covered and an explanation of the most important features of the framework. New framework users want to identify, understand and manipulate the flexible features of the framework they need, as quick as possible, without being forced to understand the detail of the whole design, but only its basic architecture (static and dynamic).

Cost-effectiveness. Unfortunately, for many frameworks, the source code of example applications is the first and only documentation provided to framework users, as they can be produced during framework development without extra effort.

Solution Provide a small but representative graded set of training examples to illustrate the framework's applicability and features, each illustrating a single new way of customization, and eventually all providing complete coverage.

The usage of hypertext links in the source code and the availability of executable code are a valuable help for understanding the examples.

Set of examples. A good set of examples can serve as a live catalogue for the basic vocabulary of the problem domain and the key features of the framework, constituting a perfect complement to all other purposes of documentation. A proper selection of examples can be very effective for illustrating the domain covered, how-to-use the framework, and revealing some design internals.

Examples. Examples play a key role in framework documentation as they make a framework more concrete, they help on understanding the flow of control, and are easier to understand than design abstractions, although less general.

The source code of example applications constructed using the framework is often the first documentation provided to application developers. The usage of hypertext links in the source code and the availability of executable code are a valuable help for understanding the examples.

The cost of producing examples to deliver with the framework is usually reduced, if well planned, as the examples can also be used to drive the development, to verify the real reusability of the framework, and to help on documenting the framework.

The examples must be used to show what the framework is good for, and not for showing how to use the framework, nor for explaining how the framework is designed.

Because good examples help new users on getting started fast, the study of working examples is a nice and motivating way of learning a framework, and help drive the learning of the framework to the points of most interest to users, thus making the learning more effective.

Examples The documentation of many successful frameworks provides a lot of examples, which make them easier to understand, to use, and to extend [16]. It was observed that the most typical documentation of successful frameworks includes examples that work right “out-of-the-box”. Some examples are: MVC [7], ET++ [13], and Java Swing [14].

JUnit. Figure 6. presents an extract from the article “Test Infected: Programmers Love Writing Tests” [15] that uses and describes an example named MoneyTest provided with JUnit.

Example

As you read, pay attention to the interplay of the code and the tests. The style here is to write a few lines of code, then a test that should run, or even better, to write a test that won't run, then write the code that will make it run.

The program we write will solve the problem of representing arithmetic with multiple currencies. Arithmetic between single currencies is trivial, you can just add the two amounts. Simple numbers suffice. You can ignore the presence of currencies altogether.

Things get more interesting once multiple currencies are involved. You cannot just convert one currency into another for doing arithmetic since there is no single conversion rate- you may need to compare the value of a portfolio at yesterday's rate and today's rate.

Let's start simple and define a class [Money](#) to represent a value in a single currency. We represent the amount by a simple int. To get full accuracy you would probably use double or java.math.BigDecimal to store arbitrary-precision signed decimal numbers. We represent a currency as a string holding the ISO three letter abbreviation (USD, CHF, etc.). In more complex implementations, currency might deserve its own object.

```

class Money {
    private int fAmount;
    private String fCurrency;

    public Money(int amount, String currency) {
        fAmount= amount;
        fCurrency= currency;
    }

    public int amount() {
        return fAmount;
    }

    public String currency() {

```

Figure 5. MoneyTest: an example provided with JUnit.

Swing. Swing provides a rich set of graded examples, very helpful for both new and experienced users on better evaluating the applicability and feasibility of the framework regarding the specific needs of the application at hands. Figure 6. shows part of a Swing tutorial fully based on examples.

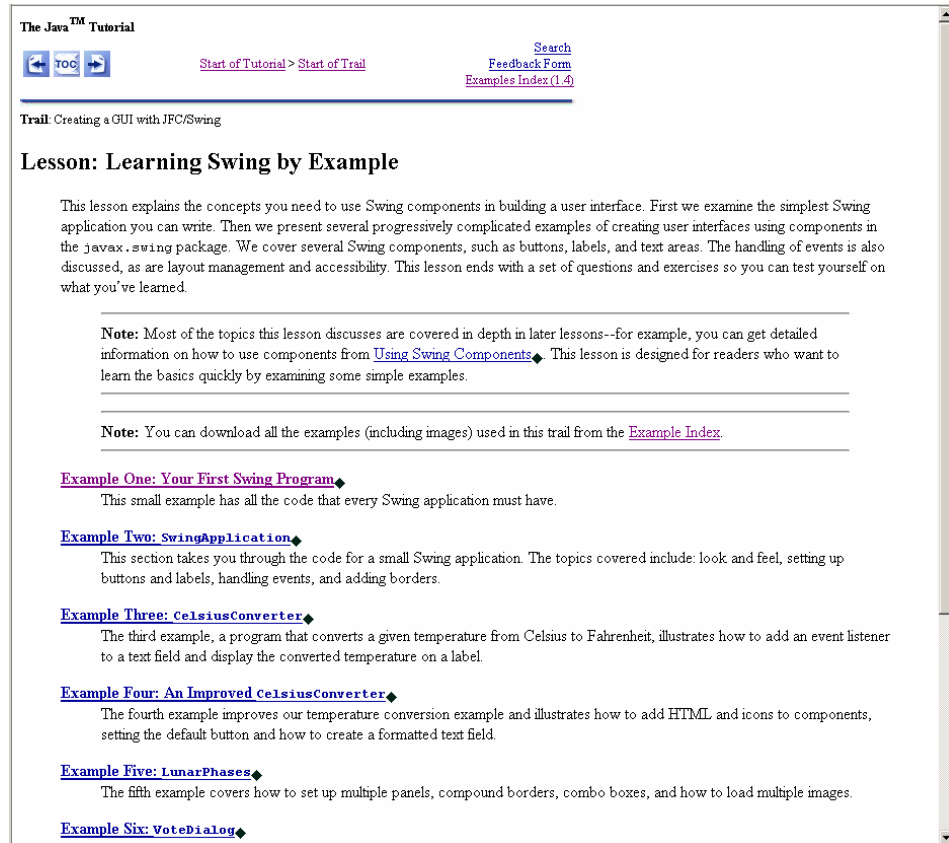


Figure 6. An example-based tutorial provided with Swing.

Consequences Providing a good set of graded examples for a framework is usually not expensive, useful during framework development and testing, and helps both new and experienced users on quickly evaluating what can be done with the framework and also reveals a little about how to use it. Despite their value, they are however not sufficient to completely document a framework.

Therefore, include in the examples references to the *cookbook and recipes* that explain the *customizable points* used in the example, optionally referencing information about the *design internals* and *source code*.

Credits The authors would like to thank our shepherd Uwe Zdun for the valuable comments and feedback provided during the shepherding of these patterns.

References

- [1] Aguiar, A., and David, G. (2005). Patterns for Documenting Frameworks – Part I. In Proceedings of VikingPLoP’2005, Helsinki.
- [2] Aguiar, A. (2003). A minimalist approach to framework documentation. PhD thesis, Faculdade de Engenharia da Universidade do Porto.
- [3] Hargis, G. (2004). Developing quality technical information. Prentice-Hall, 2nd edition.
- [4] Alexander, C., Ishikawa, S., and Silverstein, M. (1977). A Pattern Language. Oxford University Press.
- [5] Krasner, G. E. and Pope, S. T. (1988). A cookbook for using the model-view-controller user interface paradigm in smalltalk-80. Journal of Object-Oriented Programming.
- [6] Pree, W. (1995). Design Patterns for Object-Oriented Software Development. Addison-Wesley / ACM Press.
- [7] Johnson, R. (1992). Documenting frameworks using patterns. In Paepcke, A., editor, OOPSLA’92 Conference Proceedings, pages 63–76. ACM Press.
- [8] Lajoie, R. and Keller, R. K. (1995). Design and reuse in object-oriented frameworks: Patterns, contracts and motifs in concert, pages 295–312. World Scientific Publishing, Singapore. World Scientific.
- [9] Froehlich, G., Hoover, H. J., Liu, L., and Sorenson, P. G. (1997). Hooking into object-oriented application frameworks. In International Conference on Software Engineering, pages 491–501.
- [10] Apple Computer (1986). MacApp Programmer’s Guide. Apple Computer.
- [11] Beck, K. and Gamma, E. (2003b). JUnit: Cookbook. Available from <http://junit.sourceforge.net/doc/cookbook/cookbook.htm>.
- [12] Eckstein, R., Loy, M., and Wood, D. (1998). Java Swing. O’Reilly & Associates, Inc.
- [13] Weinand, A., Gamma, E., and Marty, R. (1989). Design and implementation of ET++, a seamless object-oriented application framework. Structured Programming, 10(2).
- [14] Gosling, J., Joy, B., and Steele, Jr., G. L. (1996). The Java Language Specification. Addison-Wesley. Also available online at URL <http://java.sun.com/docs/books/jls/>.
- [15] Beck, K. and Gamma, E. (2003c). JUnit: Test infected: Programmers love writing tests. Available from <http://junit.sourceforge.net/doc/testinfected/testing.htm>.
- [16] Hansen, T. (1997). Development of successful object-oriented frameworks. In Addendum to the 1997 ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (Addendum), pages 115–119. ACM Press.