A Minimalist Approach to Framework Documentation

Ademar Aguiar and Gabriel David
Faculdade de Engenharia da Universidade do Porto,
Rua Dr. Roberto Frias s/n, 4200-465 Porto, Portugal
{aaguiar,gtd}@fe.up.pt

Abstract. Good quality documentation is an important prerequisite for the effective reuse of object-oriented frameworks. To satisfy the needs of different audiences, framework documentation must integrate several kinds of documents and contents, thus resulting hard, costly and tiresome to produce when not supported by appropriate tools and methods. This research proposes a documenting approach for frameworks that aims to be simple and economic to adopt. It reuses existing documentation styles, techniques and tools and combines them in a way that follows the design principles of minimalist instruction theory. The resulting documents assume the form of minimalist framework manuals, a kind of instruction manual that emphasizes the understandability and usability of a framework. In concrete, the minimalist approach proposes a documentation model, a documentation process, and a set of tools built to support the approach—XSDoc¹.

1 Introduction

Object-oriented frameworks are a powerful technique for large-scale reuse. Through design and code reuse, frameworks help developers achieve higher productivity, shorter time to-market, and improved compatibility and consistency [1,2,3].

A framework can be defined as a reusable design of an application together with an implementation [1,4,5,6,7]. As one of the most complex kinds of object-oriented products, frameworks can be particularly hard to understand by first-time users, specially if not accompanied with appropriate documentation [2,8]. Grady Booch clearly stated in [9] that "the most profoundly elegant framework will never be reused unless the cost of understanding it and then using its abstractions is lower than the programmer’s perceived cost of writing them from scratch".

This research focus on the problem of producing good quality documentation as a means to improve the understandability and usability of frameworks [10]. The next section briefly overviews this problem and introduces the minimalist instruction theory. Section 3 presents the key concepts of minimalist framework documentation, and section 4 outlines the XSDoc infrastructure aimed to support the approach. The final section discusses the results achieved and future work.

2 Motivation and Research Overview

Good quality documentation is a crucial success factor for framework reuse because it helps on the understanding of a new framework, guiding users on the customization process and explaining their design principles and details [2,7]. Among the approaches suggested for documenting frameworks, the cookbook approach [11], the patterns approach [12] and the meta-patterns approach [13] have proven to be effective in reducing the typical long learning curve.

¹ XSDoc pronounces "Extensible Soft Doc"
2.1 The Problem of Documenting Frameworks

To define and write good quality documentation for a framework is not easy, quick or pleasant to do. It is, at least, an order of magnitude more difficult than documenting object-oriented applications or class libraries, because it must cover not only a single concrete product (an application) but, instead, a tool to produce a family of many similar concrete products (a framework). To be complete, the documentation of a framework must describe the application domain covered by the framework, its purpose, how to use it, how it works, and details about its internal design, what globally may involve a large diversity of contents and many different ways of presenting them [14]. This inherent complexity results from the following reasons:

- **different audiences** use frameworks in different ways, each with their own documentation requirements: framework selectors, application developers, framework developers and developers of other frameworks;

- **different styles of documents** are used in framework documentation to provide multiple views (static, dynamic, external, internal) at different levels of abstraction (architecture, design, implementation): framework overviews, example applications, cookbooks and recipes, design patterns, use cases, contracts, design notebooks and reference manuals [14];

- **different notations** are needed to represent different kinds of contents: free text, structured text, source code, object models, images, formal specifications, etc.

2.2 Minimalist Instruction Theory

Minimalist documentation is based on the theory of minimalist instruction [15], a theory with foundations in the psychology of learning and problem solving. The minimalist instruction intends to help on the design of instruction material, so that people can learn faster and for longer. The key idea in the minimalist instruction is to minimize the obtrusiveness to the learner of training material, hence the term.

Learners in general don’t seem to appreciate to read overviews, reviews and previews of training material. Instead of reading, people seem to be more interested in action, in working on real tasks and in doing their work. To overcome these and other obstacles to learning, minimalist instruction theory is based on three values: more to do, to allow learners to start immediately training on meaningful realistic tasks; less to read, to reduce the amount of reading and other passive activity in training; and help with errors, to help making error recognition and recovery less traumatic, more pedagogical and more productive.

As a design theory, the minimalist instruction doesn’t prescribe ways of producing minimalist manuals, but instead it defines a set of design principles: to motivate people to train on real tasks and get-started fast; to present topics very briefly in the order that seems best for the reader; to support error recognition and recovery; and to try to explore readers’ prior knowledge.

2.3 Research Goals

Much of the work on framework documentation has focused more on finding ways of documenting the design and architecture of frameworks [8] rather than on exploring effective ways of describing the purpose and intended use of frameworks.
Despite the research done, there are still open issues related with framework documentation [14,16], namely: the definition of suitable methods and tools for an effective and economic production of framework documentation; and the exploration of effective ways of describing the purpose and intended use of a framework [8]. This last issue is precisely where minimalist documentation can be helpful, i.e. on improving the understandability and usability of frameworks.

The main goal of this work is to define a flexible approach capable of reducing the costs, typically high, associated with the production of high-quality framework documentation. A secondary goal is to evaluate the impact of minimalist documentation on the understandability and usability of frameworks.

3 Minimalist Framework Documentation

The minimalist approach to framework documentation integrates reuses typical document styles and techniques of framework documentation and combines them in a way that follows the design principles of minimalist instruction theory.

The resulting documentation assumes the form of a minimalist framework manual, including information about the application domain, its purpose, how-to-use it, how it works, and internal design details. Typically, minimalist manuals are considered easy to read and understand, thereby contributing for shorter learning curves on how-to-use a system, and leading to a better understanding of the systems being trained [15,17,18].

Among the experiments reported in the literature that used minimalist instruction for documenting frameworks are [18,19].

3.1 Requirements for the Approach

The best mix of document types, writing techniques and presentation styles strongly depends on the specific objectives, context and economics of the project at hands, and even on the psychological and technical characteristics of the team elements. A good approach for documenting frameworks should be able to satisfy a diversity of requirements, with a special emphasis on the following:

- **easy-to-use** by developers, so that the activity of documentation can be a means to improve development productivity and quality, instead of being considered an obstacle, as happens in many development environments;
- **flexible** enough to be easily adaptable to the needs of different projects and development environments;
- **capable of cross-referencing** different kinds of contents using simple linking mechanisms, easy to learn and use;
- **economical**, to reduce the typical high-costs associated with the production of good quality documentation.

To fulfil all these requirements, it is important to have the support of a kind of integrated content management covering the overall process of framework documentation from the initial phases of creation and integration of contents till the last phases of publishing and presentation to target audiences.

The approach informally provides guidance about what, when, and how to document. It consists on a **documentation model**, a **generic documentation process** and a **set of tools** to make it convenient to use in mainstream development environments.
3.2 Documentation Model

The documentation model is the core of the approach. It enumerates and organizes all the contents and interdependencies (part-of hierarchies, navigational links, and derivations) required to produce a minimalist framework manual. According to its nature, the overall contents can be divided in two main categories:

- **typical of framework documentation**: code examples, recipes and cookbooks, design patterns, framework overviews, reference manuals, design notebooks, use cases, scenarios and contracts;
- **typical of minimalist manuals**: user tasks, usage patterns, task information contexts, error inventories, error recovery guidelines, and classification of contents according to the usage mode (training-wheels, guided-exploration or free exploration).

To be useful, all this web of contents must be presented in an appropriate manner, so that users don’t become overwhelmed or lost when using the documentation. Therefore, the overall repository of contents is organized as a virtual *n*-dimensional documentation space defined by *n* documentation aspects. Important examples of documentation aspects are: kind of audience, level of abstraction and level of information granularity. This space can then be divided in documentation layers according to the relevance of a specific content for a certain documentation aspect. The distribution of contents along the layers is configurable and is supported by meta-information, either manually annotated or automatically synthesized from the contents.

![Diagram](image)

**Fig. 1.** Example of a configuration of layers.

To illustrate the layered documentation space, it is represented in Fig. 1 a simple configuration of layers for a two-dimensional documentation space defined by the aspects abstraction level (abstract, concrete) and kind of audience (framework selector,
framework client and framework developer). This configuration defines three layers, named introduction, internal design and implementation, and considers four kinds of contents, framework overview, design pattern, example and class. This configuration tells us that: a framework overview contains information relevant to the introduction layer and all kinds of audience; an example contains information relevant to all layers and all kinds of audience; a design pattern contains information relevant for both the internal design and implementation layers, and for framework clients and framework developers; and a class only contains information relevant for the implementation layer and framework developers.

Fig. 2. An overview of the key concepts used in the documentation model.

The key concepts of the model are represented in Fig. 2. Starting from the top concept, we see that the project bundle is composed by all the physical units of both the code base and the document base. A code unit can be a source code file or an object code file, and contains several code elements, such as packages, classes, methods, or fields. On the other hand, a document unit can be a file, a database or a repository, and contains several document elements, i.e., pieces of documentation possibly represented in different notations, such as texts, models, annotations, etc. In addition, there are predefined document templates and sample instances of these templates, one for each kind of style of document to use, such as use-case template, pattern template, framework overview template, cookbook template, etc. Finally, the model contains also a documentation configuration that is used to define the documentation layers and the way elements are supposed to be filtered, transformed and formatted according to their relevance to each aspect and the layer in focus.

In order to cope with a vast diversity of documentation requirements, the implementation of the model should be extensible so that new custom styles of documents, notations, and layers can be added with a reasonable effort.
3.3 Documentation Process

The documentation process defines the roles, techniques, and activities involved in the production of the minimalist framework manuals. The roles identified are:

- **developers**, such as framework users, framework developers, and framework maintainers, which are responsible for content creation mostly during the development phase;
- **technical writers**, which are responsible to structure, guide, review and conclude the documentation;
- **documentation managers**, which are responsible for configuring and maintaining the documentation base, namely the template documents, template instances, and the filtering, transformation and formatting of documents according to the layers configured.

The production of framework documentation is closely related with the framework design and usage, so, ideally, these activities should be done side by-side, if we want to obtain documentation that is understandable, consistent, and easy-to-maintain.

After a configuration phase, the production of framework documentation starts with the **creation** of the various kinds of contents, and their cross-referencing. Upon creation, the different kinds of contents are normalized, integrated and stored in a repository from where they will be retrieved, transformed, published and presented to target audiences.

This documentation process is generic and was designed considering lightweight processes, which typically allocate very little effort for documentation, thus being very restrictive on adopting a documentation approach. Therefore, the resulting process is simple, flexible and easy to adapt to different development processes and environments, ranging from literate programming environments [20,21] to industrial integrated development environments. In the next section are presented the set of tools currently being prototyped to support this minimalist approach.

4 The XSDoc documentation infrastructure

The XSDoc is an infrastructure based on XML [22] and WikiWikiWeb [23] specially designed to support the production and usage of minimalist framework manuals, and covers all the typical functionalities of a content management system. Currently, XSDoc only supports frameworks written in Java programming language, models described in UML [24], and the integration with the Eclipse IDE.

The XSDoc infrastructure is composed by one Wiki engine (XSDocWiki), a plugin for integration in the Eclipse IDE (XSDocPlugin4Eclipse), and a set of document templates, markup languages, and converters of contents to and from XML. The Fig. 3 illustrates these components as well as their interconnections.

4.1 XSDocWiki

A WikiWikiWeb, or simply a Wiki, is a very innovative and appealing collaboration tool. It can be defined as a web platform for the cooperative edition of documents, where everyone can edit any page, using a simple web browser and invoking the "Edit" option on the top, or bottom, of that page [23]. After saving, the modifications done will be uploaded immediately and made available online.
A Wiki uses a very simple markup language to support simple text formatting and a mechanism for automatic linking based on WikiNames\(^2\). Despite its simplicity the mechanism is very powerful because it works like a late-linking mechanism, thus enabling the dynamic change of the targets. In addition, other kinds of linking mechanisms can be defined using lexical conventions, such as prefixes, suffixes, and name patterns, in general.

The XSDocWiki engine is the main component of the XSDoc infrastructure. It was developed using the VeryQuickWiki engine [25] as a starting base and then extended with several features to support the edition and visualization of minimalist framework manuals, namely the support for processing Java sources, UML diagrams, XML documents, version control systems, a plugin mechanism for adding new types of documents, and a few minimalist controls. The resulting Wiki supports not only the automatic linking between Wiki pages, but also between informal documents, structured documents, source code programs in Java language, and models in UML, using predefined naming conventions that are very easy to learn and use.

With the plugin mechanism, the support for each new style of document (use-case, example, cookbook, pattern, etc) can be added on the fly. A XSDoc plugin includes: a document-template; a set of converters to map that style of documents to and from XML, which can be written in a scripting language (XSL, Javascript, Python, etc.); a

---

\(^2\) WikiName is a Wiki name because it JoinsCapitalizedWords, which is AnotherWikiName
declaration of which elements may contain Wiki text, so that they can be analyzed and
their links connected; and some lexical rules to use during the automatic linking phase.
For example, for Java source files, it is declared that javadoc elements may contain Wiki
text, thus enabling in javadoc comments the usage of Wiki links to any Wiki page, be
it another Java source file, an UML file or a document file, structured or not.

So configured, the XSDocWiki promotes the cooperation of technical and non-
technical people on an incremental edition and revision of software documents, ensuring
an high availability of contents (always online), and only requires a simple web browser,
a tool currently very easy to integrate in a vast majority of development environments.

4.2 XML Converters and Presentation Processors

As most of the contents can be comfortably edited and linked using the Wiki, most of
the documentation contents will reside on Wiki pages stored in a file system, a version
control system, or a database.

However, Java source code programs and UML diagrams need special processing as
they must be converted from their original format to XML using XSL transformers [26],
respectively using JavaML [27], SVG [28] and XMI [29] vocabularies.

Before being published and presented, the contents must then be filtered and for-
matted accordingly. XSDoc is able to output HTML files for online browsing, and PDF
files for high-quality printing.

4.3 Integration Mechanisms

The components of XSDoc are closely integrated, both in terms of functionalities and
in terms of the information they exchange. The functional integration of the Wiki with
the converters and processors is done within the Wiki and its specific extensions.

In terms of the information exchanged between the tools, the integration is achieved
through the use of text files and XML files. A markup language (XSDocML\textsuperscript{3}) is also
used internally to normalize all the contents in a unique schema, when necessary.

One of the goals of the minimalist approach is its seamlessly integration in contem-
porary development environments. We think that the combined use of XML and Wiki
makes this integration successful in almost every industrial development environment
with the cost of development of small configurations, considering that XML is widely
supported everywhere and the Wiki engine only needs a browser to run.

The integration of the XSDoc infrastructure with IDE’s is achieved through the
development of specific plugins. The plugin should enable the use of a web browser
through which the XSDocWiki can be accessed, and to provide a communication link
between the IDE and the XSDocWiki, to enable their interoperation.

Much tighter integration of the XSDoc infrastructure in a development environment
can be done with recent IDE’s, such as Borland’s Together or IBM’s Eclipse, which
enable in the same environment a synchronized edition of all kinds of contents: source
code, UML models, and XSDocWiki documents.

\textsuperscript{3} XSDoc Markup Language
4.4 Benefits of XSDoc

During development, it is typical to switch between the tasks of developing code (edit-compile-test), the task of browsing documentation, and the task of writing documentation, if done.

Present IDE’s already integrate in a same environment the tasks of developing code, and some of them also enable the browsing of documentation inside the IDE. However, the support for writing documentation inside an IDE is usually very small, consisting only on enabling the writing of documentation in the same files as the source code, using Javadoc comments for example, or specific forms to introduce the Javadoc comments.

But when we need to write documentation at a higher level of abstraction than the source code files enable, such as documenting a pattern instantiation, or describing an architecture, we need to jump out of the IDE and edit the file independently. Worst, if we need to cross-reference the contents of this pattern instantiation document with some source code elements, be it a class, a method or a field, we need to copy-paste the contents from one file to the other (the most usual and easier) or instead define a static cross-reference between the contents. In any case, sooner or later, the document and the source code will become incoherent, because code changes very fast.

The most economic alternative to avoid incoherence between documentation and code is not to write documentation during development, but only at the end; another alternative is to use a literate programming philosophy, but in this case we need to move out from the most popular IDE’s and leave their powerful features that help us improve development productivity. XSDoc is another alternative to solve this problem.

With XSDoc integrated in an IDE, the developer have access to a web browser from where he can use the XSDocWiki. When documenting, the developer creates new pages, writes documents, possibly using predefined templates, uses copy-paste and drag-and-drop IDE features, browses the resources, both documents and source code, and defines links to other pages or special contents, like Java source code or UML diagrams, using predefined tags and linking mechanisms.

As an example, to document the instantiation of the Command pattern by the class TestCase requires the writing of the text shown in Fig.4(a), which produces the result represented in Fig.4(b). Any change on the code will be automatically reflected in the documentation, when the page is refreshed by the browser.

5 Conclusions

Good quality framework documentation helps users to understand the purpose of a framework, to learn how to customize it, and to learn its internal design details. A lot of work already exists on ways of documenting the design and architecture of frameworks, but there are still open issues.

Inspired by the minimalist instruction theory, this research proposes a new approach (model, process and tools) to produce minimalist framework manuals. Simplicity, low cost, and easy-to-use by all the elements of the development team, specially the programmers, are some of the intended qualities of such approach. To make convenient the practical adoption of the approach, a set of tools called XSDoc infrastructure is provided, combining a Wiki engine, document processing with XML technology, and integration in a popular IDE.
This document exemplifies how a pattern instantiation can be documented using the [XSDoc] tools.

The class TestCase instantiates the CommandPattern.

The class TestCase plays the role of the participant [Command] and the method run() implements the responsibility defined by Execute().

The resulting method run() of TestCase is presented below:

```java
public void run(TestResult result) {
    result.run(this);
}
```

From the resulting work, it can be concluded that the use of the XSDoc infrastructure integrated in an IDE such as Eclipse, can significantly reduce the effort typically needed to document a framework, as it can combine the simplicity, easiness and versatility of the collaborative document edition in the Wiki, with the well-known qualities of XML technology in terms of integration, processing and presentation of information.

Obviously, a Wiki engine adapted to the edition of XML documents can be considered by many as something more difficult to use than a typical Wiki, or a Wiki can be considered a poor XML editor when compared to a typical XML editor. Anyway, the combination of both technologies result in a very attractive infrastructure, whose best qualities can be summarized as: easy to integrate in a framework development environment; easy to use by any element of framework project team (technical or not); promotes the participation of all the team elements in the documentation process; improves the communication between the team elements; provides an easy access, revision and incremental evolution of the documentation; and finally, enables a smooth integration of contents in a controlled and structured way, informally, while preserving the information in an universal format, the XML format.

In future work, the XSDoc tools will be improved with more minimalist features (zoom, exploration mode, error recovery, extensive search) and the minimalist manuals for JUnit and JHotDraw will be concluded and their impact in terms of usability and understandability will be evaluated in comparison with the original framework docu-
mentation. Then, new plugins for integration with other popular IDE’s will be developed and other popular Wiki engines will be supported.

References