A Framework for Communicability of Software Documentation

Pankaj Kamthan
Concordia University, Canada

INTRODUCTION

The role of communication is central to any software development. The documentation forms the *message carrier* within the communication infrastructure of a software project.

As software development processes shift from predictive to adaptive environments and serve an ever more hardware diverse demographic, new communication challenges arise. For example, an engineer may want to be able to remotely author a document in a shell environment without the need of any special purpose software, port it to different computer architectures, and provide different views of it to users without making modifications to the original. However, the current state of affairs of software documentation is inadequate to respond to such expectations.

In this article, we take the position that the ability of documents to be able to communicate at all levels intrinsically depends upon their *representation*. The rest of the article proceeds as follows. We first outline the background necessary for later discussion. This is followed by a proposal for a quality-based framework for representing software documentation in descriptive markup and application to agile software documentation. Next, challenges and avenues for future research are outlined. Finally, concluding remarks are given.

BACKGROUND

Since the origins of software, and subsequently the recognition of software engineering as a discipline, documentation has had an important role to play. The use of documentation in software has a long and rich history (Furuta, Scofield, & Shaw, 1982; Goldfarb, 1981; Knuth, 1992).

There are various means that have been used for expressing software documentation. The documents could for example be expressed in structured natural language text (mimicking typewriting); Rich Text Format (RTF) and its implementations such as Microsoft Word; Hypertext Markup Language (HTML), which supports multiple language characters and symbols that can reach a broad demographic worldwide, incorporates features of print publishing, and supports hyperlinking; the Portable Document Format (PDF); and TEX/LATEX and their variations that are oriented to mathematical typesetting. However, these traditional means suffer from one or more of the following limitations: they are proprietary and can only be authored or rendered by a proprietary software; the focus is not on software engineering but other disciplines such as generic office or scientific use; and the focus is mainly on the presentation or processing rather than on representation.

Descriptive markup (Goldfarb, 1981) is based on a rich model of text known as the ordered hierarchy of content objects (OHCO) (Coombs, Renear, & DeRose, 1987; DeRose, Durand, Mylonas, & Renear, 1997) that lends a hierarchical structure to documents. The Standard Generalized Markup Language (SGML) and its simplification the Extensible Markup Language (XML) are exemplary of descriptive markup. SGML/XML are both *meta-markup* mechanisms that lend a suitable basis for a concrete serialization syntax for expressing information in a software document. They define a document in terms of its OHCO structure with mnemonic names, usually inspired by the domain being addressed, for the content objects of the data. There is a large and increasing base of markup languages based on SGML/XML.

The focus in this article is primarily on XML. Indeed, the use of XML for software process documents has been proposed (Clements et al., 2002; Mundle, 2001). The DocBook/XML markup language has been deployed for software user documentation. These efforts, however, are oriented towards technology rather than descriptive markup or communicability (or quality in general); they do not provide comparisons with other means of representations, and they do not include details of challenges posed during document engineering.

DESCRIPTIVE MARKUP AND REPRESENTATION OF SOFTWARE DOCUMENTATION

We look at a software document from two viewpoints, namely that of a *producer* and that of a *consumer*. Based on that, the representation requirements that we consider pertinent for software documentation are the following:

- **Communicability Concerns for a Document Producer:** A provider, who is responsible for both internal

and external documentation, could be interested in any of the following aspects: be able to express the domain under consideration well; have the flexibility of authoring and serving documents in different modalities; readily move documents between computing environments and over networks; easily manage the collection of documents, particularly as they scale (grow in number).

- **Communicability Concerns for a Document Consumer.** A consumer, whose main concern is external documentation, could be interested in any of the following aspects: access the documents on the device he/she is using that may be stand alone or connected to the Internet, and in the natural language/characters of choice; read or listen to the documents in the way he/she prefers that they should be presented; may want to simply look at the table of contents before reading further, or look up the definition of a term used in the main document. In some sense, a consumer would like a document to be "personalized."

These requirements motivated the proposal of a communicability framework, which we now discuss in detail.

A Framework for Communicability of Descriptive Markup-Based Software Documentation

The discussion of software documents and their representations that follows is based on the framework given in Table 1.

Semiotics (Nöth, 1990) is concerned with the use of symbols to convey knowledge. From a semiotics' perspective, a representation can be viewed on three interrelated levels: *syntactic, semantic,* and *pragmatic*. Our concern here is the pragmatic level, which is the practical knowledge needed to use a language for communicative purposes. Indeed, the goal of pragmatic quality is comprehension (Lindland, Sindre, & Sølvberg, 1994), and communicability is a prerequisite to that.

We acknowledge that there are time, effort, and budgetary constraints on producing a software document. We therefore include *feasibility*, a part of decision theory, as an all-encompassing factor to make the representation framework practical.

We now consider other aspects of the framework in more detail, starting with the principles underlying documents and their descriptive markup realizations.

Document Engineering Principles with a Descriptive Markup Perspective

The principles presented here are inspired by the established software engineering principles (Ghezzi, Jazayeri, & Mandrioli, 2003).

- **[DEP1] Separation of Concerns:** There are various concerns in document production and delivery, and to manage them effectively, each of these semantically different concerns need to be addressed separately. In particular, separation by parts, time, quality, and views are of special interest. SGML/XML provide means for separating structure, presentation, and logic in a document.
- **[DEP2] Abstraction:** This principle is based on the idea that it is often necessary in documents to highlight only the essentials while suppressing the details. The provision for table of contents and index of terms in a document are examples of abstraction. In descriptive markup, this could be realized by using mnemonic labels `<section>` and `<term>` to encapsulate the name of a section and a special term, respectively and

Table 1. A high level view of the elements of a framework for communicability of software documentation

Software Document		
Pragmatic goal	Communicability	Feasibility
Quality attributes of concern	Evolvability, Heterogeneity, Interoperability, Processability, Renderability, Traceability, Universality	
Engineering principles	Abstraction, Anticipation of Change, Formality, Generality, Incrementality, Modularity, Separation of Concerns	
Representation model	Descriptive Markup	