Documenting Software Using Adaptive Software Artifacts

Filipe Figueiredo Correia
Faculdade de Engenharia, Universidade do Porto, Portugal
filipe.correia@fe.up.pt

Abstract
Creating and using software documentation presents numerous challenges, namely in what concerns the expression of knowledge structures, consistency maintenance and classification. *Adaptive Software Artifacts* is a flexible approach to expressing structured contents that tackles these concerns, and that is being realized in the context of a Software Forge.

Categories and Subject Descriptors D.2 Software Engineering [*D.2.7 Distribution, Maintenance, and Enhancement*]: Documentation

Keywords documentation, wikis, modeling, knowledge

1. Research Problem and Motivation
Software developers capture knowledge as software artifacts of different kinds, from source-code, to models, to textual documents. Not only are some of them an integral part of the software being built, they are vital for team communication and to preserve knowledge for future use.

These artifacts are not created at the same time and evolve throughout the project's lifetime. They take part of the sense-making process in which the team identifies the information *patterns* (i.e., structure) that underly a given body of knowledge. The team may need to capture, share and reason about those ideas to discover how they can be structured, therefore they may first capture them as free-form contents and, only afterwards, capture them as increasingly more specialized artifacts, such as tasks, models and source-code.

On the one hand, capturing information's structure *explicitly* makes it more concrete, unambiguous and terse. On the other hand, free-form contents have the benefit of not being subject to structural constraints, which is important during exploratory work. One of the downsides of free-form contents, however, is that they don't directly support sharing information patterns between team members. Information is also not easily automatable — e.g., the cost of maintaining free-form contents is high, as keeping their consistency requires continuous review. Moreover, organizing and classifying information for efficient access is often difficult and classification schemes may also need to be constantly updated to reflect the evolving body of information.

Adaptive Software Artifacts combine the benefits of free-form and structured contents. It is an approach to make information within software development teams easier to use and evolve, especially in the context of medium-to-large projects where the amount of knowledge involved easily heightens all of these challenges.

SPLASH '13, October 26–31, 2013, Indianapolis, Indiana, USA.
Copyright is held by the owner/author(s).
ACM 978-1-4503-1995-9/13/10.
http://dx.doi.org/10.1145/2508075.2514873

2. Background and Related Work
A number of different approaches have addressed the capture of knowledge of a software project as *documentation*.

Their goals range from lowering the barrier to entry and collaboration (e.g. wikis), integration with other artifacts for added expressiveness, readability and maintainability (e.g. literate programming, code annotations, IDEs, software forges), or making it more unambiguous and concrete (e.g. modeling, semantic wikis).

Integrated Development Environments (IDEs) focus mainly on source code but try to provide a *whole* view of software artifacts and the processes that use them. **Software Forges** [7] differ from IDEs in that they are web-based and don't focus mainly on the source code. They allow to capture and integrate artifacts as diverse as wiki pages, version-controlled files, issue-tracking tickets, and milestones, among others. Their primary goal is to support open collaboration.

Modeling techniques are used by software developers to represent complex topics in simpler terms, focusing on capturing only their relevant aspects. Models allow to represent information with a degree of rigor and objectivity that free-form contents cannot, but modeling tools impose constraints that don't always play in the user's favor. For this reason developers often resource to *lighter* approaches, like textual documents or drawings. These approaches are especially popular for tasks of an exploratory or creative nature, when the importance of being able to record incomplete and/or non-structured information is higher that rigor. The research on **Flexible Modeling Tools** tries to fill this gap [6], by combining the benefits of free-form and model-based contents, allowing users to trade precision for flexibility whenever the occasion calls for it.

3. Approach
This approach is based on the notion of *Adaptive Software Artifacts*. A plugin was developed for the Trac Software Forge — instead of having at their disposal a finite set of software artifact types (wiki pages, tickets and milestones, to name a few), Trac users are able to create their own *types* of artifact, with a completely custom-tailored structure that fits their specific project's needs. These artifacts are *adaptive* in the sense that their attributes and relations with other artifacts are not established *a priori* and can be freely evolved by the users.

They are not bound to strict constraints like other structured artifacts usually are.

The benefits of this approach over the traditional dichotomy between *structured* and *free-form* contents extend beyond the support to expressing *ad hoc* knowledge structures explicitly. It allows the consistency of the contents towards their expected structure to be automatically assessed, and access to the wiki contents can be supported by a classification scheme that is dynamically built from the tight integration between the wiki and the *adaptive artifacts*.

The *Adaptive Software Artifacts* plugin uses several *flexible modeling* principles to combine the benefits of free-form and structured contents. However, the end-result of using this approach is not necessarily a model. This approach focuses on structuring information of text documents (i.e., of wiki pages) into smaller and meaningful *elements*, on an as-needed basis. The result is the creation of instances (artifacts) and model elements (artifact types), which sets this solution apart from other *flexible modeling* approaches, that focus on the model and meta-model levels. Moreover, the main goal is not to represent this information as diagrams, or to directly play a part in the creation of executable artifacts, but to support structuring and organizing textual contents.

Literate Programming and it's derivatives (e.g., code annotations) also allow to combine source-code with textual descriptions, but they don't delve into structuring and organizing these textual contents. Comparatively, *Adaptive Software Artifacts* encourage and leverage the creation of semantically richer documentation.

4. Contributions and Results

The Adaptive Software Artifacts plugin is possibly the most tangible part of this research from an engineering standpoint, but it is not the only one. This work comprises:

- A *software-forge* supporting *adaptive software artifacts*;
- A design patterns catalog of best practices for building a system that supports this approach.
- An experiment, conducted in an academic setting, and a case study conducted in an industrial setting, both with the goal of validating the approach.

Weaki is a wiki engine developed in the context of this work, supporting the incremental capture and evolution of structured wiki pages [2]. The lessons learned from *Weaki* are being used to extend the Trac software forge to support the notion of *adaptive software artifacts*[1].

The implementation of the plugin can be regarded as a reference architecture but, more than specific implementations, the goal of this work is to address the key principles that underlie the approach. Such is the purpose of the design patterns catalog mentioned above. These patterns were mined from existing literature and tools, and sometimes driven by the development of the plugin itself. The catalog already includes patterns of maintaining the consistency of documentation artifacts [3], the classification and indexing of contents [1], the evolution of data and meta-data in systems using the Adaptive Object-Model architectural pattern [4], and of object-oriented meta-architectures [5].

[1] The current implementation is a plugin for Trac, and can be found in the address *https://github.com/filipefigcorreia/TracAdaptiveSoftwareArtifacts*

Furthermore, two user studies are being conducted to validate the approach. An experiment was performed with the participation of a sample of 43 students divided in two groups — *control* and *experimental*, and is expected to provide preliminary evidence of the approach's benefits and liabilities. It consisted of a programming exercise, comprised by a series of simple tasks that required the use of a project's documentation, built using *adaptive software artifacts*. The plugin was instrumented to collect usage data, and the participants were asked to answer a questionnaire about their use of the tool. While some conclusions can be directly derived from the usage data, others are intrinsically subjective and will be based entirely on the results of the questionnaires. At the time of writing of this paper the data is still being subject of analysis, but it is expected to help answering (among other specific issues) if contents are regarded as more *precise*, *concise*, *easier to understand*, *easier to find* and more *consistent*, when using this approach.

If the results from this experiment are encouraging, they may help to motivate the use of the plugin in the industry. Two software companies have shown interest in providing feedback on the plugin, and one in particular has shown the will to take part of a case study. This case study is expected to provide some evidence regarding the authoring of contents and may reinforce the results of the experiment.

References

[1] F. F. Correia and A. Aguiar. Patterns of information classification. In *Proceedings of the 18th Conference on Pattern Languages of Programs*, Portland, OR, USA, Oct. 2011.

[2] F. F. Correia, H. S. Ferreira, N. Flores, and A. Aguiar. Incremental knowledge acquisition in software development using a Weakly-Typed wiki. In *Proceedings of the 5th International Symposium on Wikis and Open Collaboration*, Orlando, FL, USA, Oct. 2009.

[3] F. F. Correia, H. S. Ferreira, N. Flores, and A. Aguiar. Patterns for consistent software documentation. In *Proceedings of the Pattern Languages of Programs*, Chicago, IL, USA, Aug. 2009.

[4] H. S. Ferreira, F. F. Correia, and L. Welicki. Patterns for data and metadata evolution in adaptive Object-Models. In *Proceedings of the 15th Conference on Pattern Languages of Programs*, Nashville, TN, USA, Oct. 2008. ACM.

[5] H. S. Ferreira, F. F. Correia, J. Yoder, and A. Aguiar. Core patterns of object-oriented meta-architectures. In *Proceedings of the 17th Conference on Pattern Languages of Programs*, Reno, NV, USA, Oct. 2010.

[6] H. Ossher, A. van der Hoek, M. Storey, J. Grundy, R. Bellamy, and M. Petre. Workshop on flexible modeling tools: FlexiTools 2011. In *2011 33rd International Conference on Software Engineering (ICSE)*, pages 1192—1193. IEEE, May 2011.

[7] D. Riehle, J. Ellenberger, T. Menahem, B. Mikhailovski, Y. Natchetoi, B. Naveh, and T. Odenwald. Open collaboration within corporations using software forges. *IEEE Softw.*, 26(2): 52–58, 2009.