

# Supporting the Evolution of Software Knowledge with Adaptive Software Artifacts

Filipe Figueiredo Correia

Faculdade de Engenharia  
Universidade do Porto  
filipe.correia@fe.up.pt

## Abstract

The knowledge of software developers materializes itself as software artifacts, that may be seen at two different levels (information and structure), which are difficult to change independently from each other. This work explores how the expression of software knowledge using adaptive software techniques, may support the creation of adaptive software artifacts, to improve the effectiveness of capturing knowledge under constant evolution.

Some work already exists in the context of the Weaki Wiki, which will be extended into a full environment supporting the creation and evolution of software artifacts beyond their initial form. We intend to validate this work experimentally.

**Categories and Subject Descriptors** D.2.6 [*Programming Environments*]: Integrated environments; D.2.7 [*Distribution, Maintenance, and Enhancement*]: Documentation

**General Terms** Design, Documentation, Experimentation

**Keywords** knowledge capture, software artifacts, adaptive software, development tools, software forges

## 1. Introduction

Software artifacts are commonly captured as standalone files, sometimes managed by advanced integrated development environments, but web-based systems, such as wikis and software forges, have been increasing their use in recent years due to their collaborative nature and ease of use.

Regardless of the medium, artifacts are a form of captured (i.e., recorded) knowledge, but while software knowledge evolves freely in the minds of the project's team members, artifacts are not always as easy to adapt. Consider a development team, that starts by recording software requirements as documents. Although initially informal, this information will need to end up distilled into a concrete software system. With this goal in mind, developers choose to create additional artifacts — such as user-stories, models and design documentation — giving information a more concrete and unambiguous form. When creating a domain model of the system, developers resource to the descriptive documents that already capture the knowledge they need, even if in an unstructured way. They create these new models completely from scratch rather than reusing or adapting the actual documents. This implies that the concepts and entities that they describe aren't causally connected to the ele-

ments of the newly created models, which may stem concerns like consistency and traceability.

The goal of this work is to give new capabilities to artifacts supported by Web-based environments. In concrete, we will leverage existing techniques for adaptive software to allow developers to evolve existing artifacts, and to capture new artifacts by reusing and molding the information of existing ones, so that the causal connections between them are maintained.

## 2. Evolving Software Knowledge and Artifacts

Developers are *knowledge-workers*, as their activity revolves around acquiring, processing and capturing knowledge, with the ultimate goal of obtaining instructions to be executed by a computer [7].

Two levels may be considered within software artifacts: the *information* and the *meta-information*, also referred to as *structure*. While the former refers to the particular subject that the artifact intends to represent, the later describes the information itself, contextualizing it, and conferring it additional semantics. While some structure is not changeable, artifacts frequently allow the creation of additional structural elements, that are open to being authored. The process of capturing such structures is one of carefully organizing and classifying knowledge.

The allowed expressiveness is an important factor when choosing which type of artifact to use. Document artifacts, for example, allow one to capture virtually any topic, as their structure is domain-agnostic, enforcing only a layout form. For this reason, richer artifacts are usually included in documentation, for a better balance, between flexibility and expressiveness [1].

Moreover, software systems can hardly be seen as immutable entities. The difficulty in identifying all the requirements right from the start is well known in industrial environments. Although this is sometimes blamed upon the lack of the stakeholders' knowledge [6], it's inevitable that the knowledge of stakeholders and developers will change throughout the project, which implies that artifacts usually need to be adapted to new understandings. Our key concern is that, although knowledge may evolve, software artifacts are usually more difficult to adapt accordingly, especially when structural changes are needed.

## 3. Techniques for Adaptive Software

Adaptable systems are those that can be efficiently molded according to changed circumstances. One way of creating an adaptive system is through a meta-architecture: one in which the program manipulates itself as if it were data. Meta-architectures usually describe the system's domain, or part of it, by establishing different levels of (meta-)data, that comply to each other.

Several approaches to the creation of adaptable systems exist. Dynamic approaches, like those using the Adaptive Object-Model (AOM) pattern [8], allow systems to be adapted at runtime. The system interprets a high-level description of its domain, and adapts its behavior to any changes introduced to that description.

An alternative to dynamic approaches is Generative Programming: the high-level description of a system is used to automatically create executable code, or a code skeleton that will be further completed by the developers. Adaptability is thus introduced at compile-time, requiring a full generation/compilation cycle.

#### 4. Research Problem and Goals

Software development tools support the evolution of artifacts to some extent, but they normally assume that artifacts have a fixed format. Artifacts that could be easily evolved throughout the project's lifetime, in what concerns both their contents and structure would bring benefits to knowledge capture activities. Information usually first appears as informal, and only gradually becomes more structured, and captured into richer artifacts. A classic example is how requirements are often first captured as descriptive documents, and only afterwards materialize as models, issue-tracking tickets, and source-code, among others.

The existing techniques for adaptive software allow developers to build software systems supporting domain adaptability. The same techniques may help us build development tools supporting software artifacts that are more adaptable.

**We believe that supporting the expression of software knowledge through adaptive software artifacts will improve the effectiveness of capturing knowledge under evolution.**

Our objective is to find an approach, to be supported by software development tools, that allows software artifacts to be easily changed, in what concerns both their contents and their structure. With this purpose in mind, we will:

- Identify the difficulties in the creation and evolution of software artifacts from a knowledge capture standpoint;
- Build a theory on how these needs may be fulfilled;
- Create new tools, or extend existing ones, that better support the capture and evolution of captured software knowledge.

#### 5. Research Approach

Research in software engineering needs to go beyond concrete implementations and tools and consider the human side of software development, and all of the context that underlies it. Given the amount of possibilities that this entails, research strategies usually encompass different methods, which are combined to improve the knowledge on the subject.

In the course of this work we will conduct quasi-experiments on an academic environment, and case-studies on an industrial environment. In the first phase, both will be done with an exploratory aim, to collect new insights on the problems underlying the creation and evolution of software artifacts. Later, the research will take a confirmatory goal: industrial case-studies will provide feedback and drive the development of tools; and together with the academic quasi-experiments they will be used to assess our claims.

To explore the phenomena that artifacts go through during software development, we will try to answer the following questions:

- Which difficulties are found by developers during the capture and evolution of contents?
- What are the effects of such difficulties?
- How hindered is expressiveness when artifacts need to evolve?

To answer these questions, and again later to assess the results, a set of metrics will be used:

- How much *technical debt* and *software aging* exist?
- How *long* do capture activities take?
- How *expressive* can developers be?

#### 6. Current Work

Wikis are nowadays very popular in software development. They are an effective approach to collaborative knowledge capture, and used extensively to support software documentation artifacts.

*Weaki* is a wiki engine developed in the context of this doctoral work, which supports the incremental capture and evolution of structured document artifacts [2]. For now, its development has focused on dealing with evolving document structures, and increasing awareness by the users towards the structure of contents. Although it possesses no specific features for software development at this time, our goal is to apply it to this domain, allowing the integration of some of the most common types of software artifacts, and supporting evolution beyond their original form.

We plan to support its development using an AOM design. A framework for the development of AOM-based systems has been under development, and will be used for this purpose [5].

Patterns have also been mined from existing literature and tools, and will also be used during development [3, 4].

#### 7. Research Plan

The remaining of this research work will encompass these phases:

- **Identify further needs of knowledge capture and evolution in the context of software projects.** The recognition of these needs will be made through literature review, case-studies and quasi-experiments. The approaches currently used to handle them will be documented as patterns;
- **Create a prototype environment.** Taking the development of *Weaki* further, an environment will be created for managing software artifacts, that will allow their expression, and their evolution beyond an initial structure;
- **Validate the approach.** Quasi-experiments and case-studies will be used to validate the use of adaptive software artifacts and their benefits in a context of evolving software knowledge;
- **Document and consolidate the obtained results.** The results previously obtained will be analyzed and documented.

#### References

- [1] A. Aguiar and G. David. WikiWiki weaving heterogeneous software artifacts. In *Proceedings of the 2005 international symposium on Wikis*, pages 67–74, San Diego, California, 2005. ACM.
- [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, Florida, USA, Oct. 2009. ACM.
- [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, Illinois, 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 Pattern Languages of Programs*, Nashville, Tennessee, USA, Oct. 2008. ACM.
- [5] H. S. Ferreira, F. F. Correia, and A. Aguiar. Design for an adaptive Object-Model framework: An overview. In *4th Workshop on Models@run.time at MODELS 09*, pages 71–80, Oct. 2009. URL [http://ceur-ws.org/Vol-509/MRT09\\_proceedings.pdf](http://ceur-ws.org/Vol-509/MRT09_proceedings.pdf).
- [6] R. Garud, S. Jain, and P. Tuertscher. Incomplete by design and designing for incompleteness. In *Design Requirements Engineering: A Ten-Year Perspective*, pages 137–156. Springer, 2009.
- [7] P. N. Robillard. The role of knowledge in software development. *Communications of the ACM*, 42(1):87–92, 1999.
- [8] J. W. Yoder, F. Balaguer, and R. Johnson. Architecture and design of adaptive object-models. *ACM SIG-PLAN Notices*, 36(12):50–60, Dec. 2001.