Using Structured Wikis in Software Engineering

Lisa Dyer
Lombardi Software, Inc.
4516 Seton Center Pkwy
Austin, TX 78759
+1-512-692-1244
lisa.dyer@gmail.com

Anne Gentle
Advanced Solutions International
11044 Research Blvd.
Austin, TX 78759
+1-512-491-0550
annegentle@justwriteclick.com

ABSTRACT

In this paper, we describe methods for creating structured wikis and wiki pages using the Darwin Information Typing Architecture (DITA) standard to enable software development processes.

DITA is a well-established OASIS standard for creating standard information types and domain-specific markup vocabularies with XML. The DITA OASIS standard builds content reuse into the authoring process, defining an XML architecture for designing, writing, managing, and publishing many kinds of information in print and on the Web.

Merging DITA's strong semantic markup, reusability, and output capability with the flexibility, collaboration, and ease of authoring of a wiki offers product development teams a viable environment for efficient and practical information exchange and delivery. Software engineering and development processes can be streamlined and efficient when a highly collaborative and motivated staff has the tools to create software, from the research and design phases, through coding, testing, documenting, and delivering high-quality software products.

Categories and Subject Descriptors

H.4.3 [**Communications Applications**]: Information browsers.

General Terms

Management, Documentation, Performance, Standardization, Design, Verification

Keywords

DITA, Darwin Information Typing Architecture, XML, wiki, Agile, development, information development, coding, testing, quality engineering, software, user assistance, documentation, single-source publishing

1. INTRODUCTION

Wikis and open source systems often go together. They enable teams to make significant improvements to their processes and output without significant investment. However, as the systems become strategic assets, their scalability and performance become more critical. When the right systems for efficient information exchange and collaboration are provided, the hard work and dedication required for software engineering can become a passion, regardless of whether the software is open source or not.

To aid in this information exchange and collaboration, open source software now has an open source documentation standard with which to pair. When you combine software, wikis, and the DITA open standard, software engineering becomes an efficient, scalable, and collaborative process.

By using structured data in wikis, you can make innumerable gains in knowledge management, which make predictable and efficient information exchange possible. DITA offers task, concept, and reference information typing that are useful for many team members working on tasks collaboratively in software development. From design to development to documentation to testing activities, there is a valuable reuse proposition when you can use a structured, minimalist, and consistent content model in wiki pages.

This paper discusses real-world implementations of using wikis in an Agile environment in which information is structured and which includes integrated components such as issue tracking. Although this experience report in written mostly in the Agile context, it can be applied to software development processes in general.

When you talk to software developers, you quickly find that they use wikis to help them get their jobs done. Typically, software developers and designers use wikis to share project information such as meeting minutes, provide reference information about a feature, or describe to a software quality engineer or information developer how something is supposed to work.

Wikis can be unstructured and chaotic, though, and when the wiki's content is not quite trustworthy, or the content becomes overwhelmingly disorganized and hard to find, the wiki becomes less useful and does not match its full potential as a collaborative tool.

Our solution to some of these problems is to offer a structured wiki with specific topic typing that follows the suggestions set forth by the OASIS open standard for technical documentation, DITA or Darwin Information Typing Architecture. We also highlight some key features that every wiki should support.

2. INFORMATION EXCHANGE IN AN AGILE DEVELOPMENT ENVIRONMENT

The main principle of Agile development is to develop robust software rapidly with minimal expense and little investment in rapid, up-front design. “Robust” and “rapid” are manifested in the use of iterations, or short development cycles (usually 2-3 weeks), in which a particular piece of the software is developed, tested, and documented. A development cycle built on iterations allows for rapid and continuous delivery, and it provides the agility and flexibility that are missing in more traditional development methodologies. Problems are quickly discovered, and teams can either immediately correct a problem or eliminate an affected feature altogether without wasting design work that would have occurred in traditional methodologies, such as waterfall.

While documentation is not valued as highly as working code in an Agile environment, developers and testers often use wikis to get valuable information off their minds and to a Web page for all to see and benefit from that knowledge. All phases of Agile development can benefit from a centralized wiki being the one and only source of information for an Agile team.

The Design Structure Matrix (DSM) offers a way to understand how information exchange is critical to development activities and plans. Three basic systems can be represented by parallel (or concurrent), sequential (or dependent), and coupled (or interdependent) information exchange cycles. Applying a DITA content model to your wiki helps drive development by allowing for concurrent, dependent, or interdependent activity equally.

2.1 Why Store Content Source in DITA XML

Nearly all software engineering efforts involve standards, and for your wiki it makes sense to have standards. DITA can offer standard information sets that offer mapping of task, reference, and concept for each of the areas of software engineering. DITA also provides a specialization framework to capture additional data by creating organization and industry-specific elements.

By training all users of your wiki about the three types of information that are useful to put in the wiki in certain areas for certain goals, and to make use of its content reuse constructs, you can ensure that the information lives on and is trustworthy through all phases of development.

DITA XML also gives you the ability to transform the content into different output formats—from wikitext to HTML to print-quality PDF output. Your content does not have to live in wiki format only, and can live on further in the development cycle beyond just the internal audience to the external audiences such as customers, technical support, consulting, and technical implementation personnel.

Our experience strongly suggests that a fully DITA-enabled wiki offers the following benefits:

- Information developed on a wiki supports and documents the team's planning and implementation activities throughout the release cycle.
- Content must be repurposed to a lesser or greater degree by the different work groups: Quality engineering and information development groups mine design and implementation specifications to build their test and documentation plans. These two groups lend examples of necessary repurposing of the wiki content.
- On a DITA-enabled wiki, content can be structured in a highly reusable way: An API implementation goes from market requirements to development requirements, user assistance requirements, and QA requirements in a "reference" page structure.
- Centralized system of knowledge management and knowledge sharing is the wiki way.
- You can filter different views of the same information, such as technical implementation details for developers, but summaries only for directors and executives.
- The DITA constructs offer more production-quality output options via the DITA Open Toolkit, an open source project on SourceForge, including automation that can be integrated with product builds.
- Wikis by their nature give team members collaborative authoring and discussion capability across the team.

2.2 Information Typing

Training all software engineers and quality engineers to think of information as three distinct types takes little explanation because it is a logical grouping of information. Here are the basic descriptions for each DITA topic type:

Concept: Answers the question, “What is?” Provides rules and guidelines.

Task: Outlines procedures, with a single imperative command for each step, so that the reader can accomplish a goal.

Reference: Offers undisputed facts and can be used for command syntax or application programming interfaces.

Once everyone on the team understands the value of authoring based on topic types, and the ease of publishing multiple outputs, you can use templates for your wiki pages to enable the topic typing.

Requiring that information be parsed into concepts, tasks, or reference topics regardless of whether you are using DITA or not gives you these benefits:

- Collaboration and standards with team members
- Agreed-upon terminology mapped to the documents that assist in communicating about the software to be delivered
- Future matching of user documentation to what was built in the software engineering process
- Portability of the content to other formats, including print-ready documents and application help
- Standard markup for software development references such as API documentation

3. HOW WIKIS CAN ASSIST IN THE SOFTWARE DEVELOPMENT PROCESS

This section discusses the areas of software development and how structured wikis help. Each section addresses how structured wikis can be used to write documents such as software requirement specifications in a traditional environment or user

stories in an Agile environment, store test plans, and incubate and produce user documentation in online and offline formats.

A case study is also offered.

3.1 Software Design Documents

Software requirement specifications, which list the features and designs required to be part of a complete release of a software component, are replaced with user stories in Agile that are thinly sliced representations of what a user can do with a software feature.

Since a certain number of user stories are typically completed during iteration, it is important for the user story to contain only the amount of code that can be written and tested (and likely documented) in the amount of time given to iteration. By storing these user stories in a wiki, you offer the latest update online at all times and you give teammates the ability to discuss the user story on the wiki for all to see. By enforcing a topic type such as task on the user story, you can then reuse the content created by the design document for test plans and for user documentation at the end of the iteration.

Wiki pages that contain user stories also give a dashboard view of the progress on each feature or user story. Lots of discussion around a user story might indicate that the design is not settled yet. Few details in a user story might prevent it from being reused by quality engineers for testing and delay or deter information development from deciphering what information affects users based on the user story.

3.2 Software Test Plans

Quality engineers must devise a document that describes the scope of testing, their focus, and the approach they plan to take to ensure the software feature is tested to the limits and also indicate whether it passes the thresholds set for guaranteeing that the software is useful.

By storing test plans on the wiki, using appropriate topic typing such as concept and task, the team can review the test plans, look for omissions or errors, devise outer boundaries for the test, and discuss the approach, scope, and focus for the test. Whether using the Agile development method with user stories matching up to test plans or a waterfall method where the software design specification can be correlated to a set of tests, storing the tests in a wiki helps the entire team understand what will be tested to ensure the quality of the software being built.

Unit tests written by quality engineers and developers can also be stored on wiki pages for reuse across team members and as examples for others to follow.

When testing begins and defects start progressing through the issue tracking system, teams can find ways to link the two systems with either URLs to launch the issue tracking system, or by enforcing that a tracking ID is required on the correlating wiki pages that describe the expected behavior and possible exception states. Defect tracking within the structured wiki is valuable to information development and support professionals who must later inform the user about known issues or workarounds that are unavoidable.

3.3 User Assistance

User assistance is the information that a user needs to figure out how to use the product, what inputs are required and optional, and to learn by doing tasks related to their goals. User assistance can be embedded directly into the interface, a printed manual, or a searchable online or offline help system, and there are several different formats for delivering both printed and online documentation.

Information developers constantly conduct information exchange with many groups inside and outside the product engineering team throughout the release cycle, including:

- Quality engineering and quality discovery
- Education services
- Product management
- Sales engineering and technical consulting

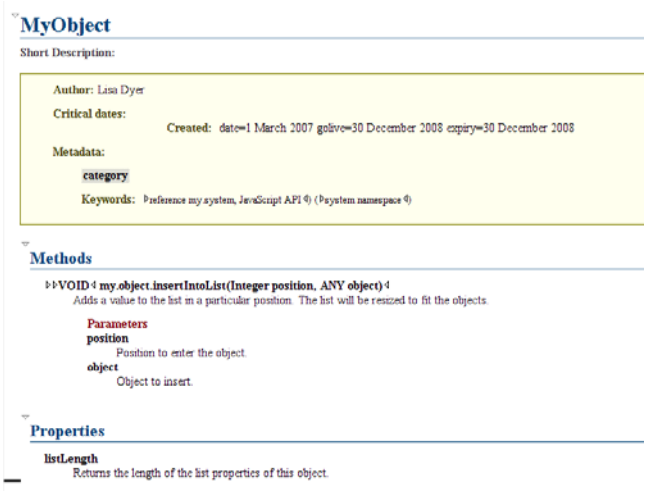
Iterating the content can be costly if efficient processes and structure are not applied. User assistance is where DITA and wikis really shine. DITA was originally created as a technical documentation model at IBM. Concept, task, and reference topics are the backbone of nearly all end-user information systems.

Plus, DITA's output transforms which come with the DITA Open Toolkit give you Eclipse help systems that integrate directly with the Eclipse Integrated Development Environment (IDE). If your product does not integrate with Eclipse, Microsoft Help and cross-platform Web Help formats are available. Also, since DITA is XML, any transformation can be written to produce the exact format of the user assistance your product needs, including embedded help.

3.4 Code Reference Information

Often developers document how an application programming interface (API) works within the code used to create the API. By using DITA XML as an information go-between, or as the storing format, the information can be semantically encoded and processed to many formats with minimal customizations. Rather than copying and pasting or using another automated mechanism, you can create a DITA specialization based on the reference topic type that will deliver exactly the API your product needs.

A JavaScript topic might render on the wiki like this:



And the DITA source for the topic might resemble the following markup (the example uses a standard DITA DTD with a custom XSL transformation to produce the wiki output):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE reference PUBLIC "-//OASIS//DTD DITA Reference//EN" "http://docs.oasis-open.org/dita/v1.1/OS/dtd/reference.dtd">
<reference id="MyObject" otherprops="bpd">
    <title>MyObject</title>
    <shortdesc/>
    <prolog>
        <author>Lisa Dyer</author>
        <critdates>
            <created date="1 March 2007" golive="30 December 2008" expiry="30 December 2008"/>
        </critdates>
        <metadata>
            <category audience="DITA authors" status="new"/>
            <keywords>
                <indexterm>reference my.system, JavaScript API</indexterm>
                <indexterm>system namespace</indexterm>
            </keywords>
        </metadata>
    </prolog>
    <refbody>
        <section id="methods">
            <title>Methods</title>
            <dl>
                <dlentry id="insertIntoList" props="VOID">
                    <dt><b><ph>VOID</ph> my.object.insertIntoList(Integer position, ANY object)</b></dt>
                    <dd>Adds a value to the list in a particular position. The list will be resized to fit the objects.</dd>
                    <dd id="params">
                        <dl>
                            <dlhead>
                                <dthd>Parameters</dthd>
                            </dlhead>
                            <dlentry id="position" otherprops="false" props="Integer">
                                <dt>position</dt>
                                <dd>Position to enter the object.</dd>
                            </dlentry>
                            <dlentry id="object" otherprops="false" props="ANY">
                                <dt>object</dt>
                                <dd>Object to insert.</dd>
                            </dlentry>
                        </dl>
                    </dd>
                </dlentry>
            </dl>
        </section>
        <section id="props">
            <title>Properties</title>
            <dl>
                <dlentry id="listLength" props="Integer">
                    <dt>listLength</dt>
                    <dd>Returns the length of the list properties of this object.</dd>
                </dlentry>
            </dl>
        </section>
    </refbody>
</reference>
```

Standard DITA attributes (id, props) are used to produce context-sensitive help for the software application (not shown).

3.5 Case Study

At Lombardi Software (a business process management solutions provider based in Austin, Texas), requirements to publish documentation updates between releases and enable users to enrich the content while retaining the ability to build a complex set of portable and embedded help formats started them down the path toward a wiki solution for their product information exchange process.

Let us study Lombardi Software's approach to wiki use throughout the software engineering process.

Their development team currently uses two wikis: One is internally accessible only, and the other externally accessible with security controlled by assigned user accounts. Their wiki system also includes a DITA-to-wiki import utility that supports key DITA constructs such as *conref* (transclusion) and *ditaval* (conditional processing of content) and preserves discussion threads on pages.

3.5.1 User Stories

At the beginning of a release, the internal wiki has user stories written in a task topic format.

Table 1. Mapping content elements to DITA elements

Content element	DITA element
Goals	`<context>`
Preconditions	`<prereq>`
Steps	`<steps>,<substeps>`

User stories are unique to Agile development practices, but other development practices can design similar use cases in software specifications.

Each user story has supportive information.

Table 2. Mapping content element to DITA elements

Content element	DITA element
Persona descriptions	`<audience>` metadata
Terminology descriptions	`<glossentry>,<indexterm>`

This initial authoring on the wiki shows the general scope and focus of a given release cycle. Any discussions of development or design decisions can be either comments on the user story wiki page itself or extremely complex explanations could be stored as wiki pages that map to DITA topic types.

Software design typically uses a reference topic type, and especially for highly structured content such as API documentation, providing a structured template is important. Documenting at the earliest possible stage on the wiki even when implementation is incomplete helps test and user assistance teams plan their efforts more correctly, and a template exposes and helps track open questions effectively.

As user stories or use cases develop as part of the iterative design process, the test and user assistance plans are created for them, linking back to the user stories and reference content.

As revisions occur, the user assistance deliverables, including embedded help, are always in the most recent state, thanks to the DITA framework which enables the multi-channel publishing process.

3.5.2 JavaScript API Documentation

JavaDoc is a standardized and predictable format for documenting Java APIs. But there is no standard "JavaScriptDoc" implementation, which makes developing and maintaining JavaScript API documentation more challenging. The following example illustrates how using DITA with wikis improved a formerly error-prone and inefficient process.

At Lombardi Software, developers and information developers document the JavaScript APIs collaboratively. The designers and developers describe the methods, properties, and other elements, and provide usage examples. Information developers provide supporting content and edit all of the content to ensure consistency and clarity. Using XSL, a subset of the content is transformed to context-sensitive help, available directly in the product interface, and the full content set is available in searchable and portable HTML and PDF formats.

Prior to offering a standard for authoring reference material related to JavaScript API, developers would document the API using a in-house, custom XML schema that only served the purposes of providing context-sensitive help within the interface. The structure of the schema was complex and confusing, difficult to maintain, and had no framework to transform the content to other outputs. As you might expect, the information developers cut and pasted from one format to another, added supporting content, and incorporated review feedback. Changes to APIs were

easily missed and cut and paste techniques were simply inefficient.

What before resided in multiple and isolated bodies of content whose states were often difficult to determine and manage, and developed in custom schemas and binary formats, is now a single body of content sourced in a standard DITA XML schema and published using an easily managed and scalable build framework.

3.6 KEY WIKI FEATURES FOR SOFTWARE ENGINEERING

Wikis have dozens of features, and not all of them are critical to software engineering processes. The following list is not offered as a complete list, but rather, a summary of some key features that facilitate information exchange and publishing processes as described in this paper:

- Paste screen shots from the clipboard
- Embed recordings for in-page viewing
- Transclusion of content from other pages (where the source is editable and transcluding instances read-only)
- Filter content based on audience needs (summaries versus details)
- Built-in workflow
- Integration with issue tracking systems
- Plug-in architecture
- Standard APIs for integration and content extraction

4. HOW TO MAKE YOUR WIKI DITA-ENABLED

Software solutions for integrating DITA with wikis are emerging, either as import and export utilities, or as DITA-aware Web applications. Even if your wiki is not storing content in the DITA XML schema—at the time of this writing, a fully DITA-enabled wiki is not yet generally available—you can implement something close to it with some wiki page scaffolding, DITA importing and exporting tools (some of which are now available as open source as listed below), and a well-defined process.

Analyze the content on your wiki and identify which types of pages occur most frequently. Map those pages to DITA topic types and create page templates that are structured like DITA XML topics.

Here are some real examples of content where the DITA model shines:

- **Terminology**: Maps to the reference topic in DITA. As features are developed, names and labels change, typically right up until the release cycle has reached the qualification stage. It is difficult to record and share evolution of terms. *Solution*: At the start of design, create a "Terminology" wiki page. Update as you go. The page is useful throughout the development cycle for the entire team. When the content reaches its qualified state, all required outputs can be generated quickly.
- **API documentation**: Maps to the reference topic in DITA. An excellent fit for DITA-enabled wikis because of the inherent structure in the content. If an API is designed well, its initial design documentation should very closely match the final documentation.
- **User stories and scenarios**: Maps to the task and concept topics in DITA. Quality engineering and information development can generate test cases and task topics from this content. Development can document unit tests.

Related DITA-based tools and solutions include:

- DITA Open Toolkit (DITA OT) http://sourceforge.net/projects/dita-ot/
- DITA to Wiki Project (DITA2wiki) https://sourceforge.net/projects/dita2wiki/
- inmedius DITA Storm (DITA-enabled Web application) http://www.inmediusdita.com/c_developer/developer.html

5. CONCLUSION

A DITA-enabled wiki environment can speed up a product development cycle by enabling the entire product development and design team to develop and exchange information as the product features evolve during sprints and then harden during the final qualification stages.

The content in a structured wiki can serve all groups involved in the software engineering process because its revision history captures discussions, implements a repeatable and predictable information structure, and is stored in a format from which all final content deliverables can be programmatically generated with semantic markup preserved.

By conducting the information exchange in a structured and standardized framework, teams can achieve a deeper understanding of the individual and shared processes and improve communication.

6. ACKNOWLEDGMENTS

Our thanks to the software developers and quality engineers who contribute to wikis, and to the information developers and business analysts who collaborate throughout the information exchange cycle.

7. REFERENCES

[1] OASIS Darwin Information Typing Architecture (DITA) standard, 2005. http://www.oasis-open.org/specs/#ditav1.1

[2] Yassine, A. and Braha, D. 2003. Complex Concurrent Engineering and the Design Structure Matrix Method. Concurrent Engineering, Vol. 11, No. 3, 165-176 (2003). SAGE Publications, Thousand Oaks, CA. DOI: 10.1177/106329303034503. http://cer.sagepub.com/cgi/content/abstract/11/3/165

[3] Dyer, L. 2008. Lombardi Wikis – a model for collaborative information development. http://www.slideshare.net/lisa.dyer/lombardi-wikis-model