INTERCONNECTING DOCUMENTATION
Harnessing the Different Powers of Current Documentation Tools in Software Development

Christian Prause[1], Julia Kuck[2], Stefan Apelt[1]
Reinhard Oppermann[1] and Armin B. Cremers[2]

[1]*Fraunhofer Institut für Angewandte Informationstechnik, Schloss Birlinghoven, St. Augustin, Germany*

[2]*Institute of Computer Science III, University of Bonn, Germany*

Keywords: Software Documentation, Software Engineering Process, CASE, Collaboration, Information Retrieval, Hypermedia, Document Management, Knowledge Management.

Abstract: Current software documentation tools (like text processors, email, documentation generators, reporting, configuration management, wikis) have different strengths in supporting the software engineering process. But one weakness they all have in common is their inability to combine the advantages of the various techniques. Integrating documentation with diverse origins would enhance the force of expression and compensate individual failings of the different techniques.
In this paper, we present a new brand of documentation utilities — exemplified by the Deudroid-system — that overcomes current problems with documentation. By processing information at negligible cost that common tools ignore, our system represents an efficient way of improving software documentation.

1 INTRODUCTION

During the development of a software product, generally a mess of unsorted documentation texts arise besides the official internal documentation. Such texts are often only loosely connected, if not without any internal references at all. These documents may contain vital project information which could get lost if the document itself gets lost.

In this context *getting lost* means that the document can get lost physically — e.g. an email being deleted — or it can get lost in the sense of being undiscoverable under a justifiable investment of time. Mailing list archives have already been identified as a repository of valuable information, but only if one is able to retrieve the relevant documents (Kirsch et al., 2006). Classical search engines for searching such repositories are only of limited help in such a situation, because the user has to actively start a search on a repository. Search engines also often only search a single[1] group of media.

We consider all kinds of asynchronous communication in text form to be documentation. So emails are not the only stand-alone documents and there are more such sources like text files or logs from configuration management. Even large units of documentation like those generated from source comments using documentation generators can be considered *stand-alone* compared to the entirety of available documents.

One might consider enforcing strict documentation guidelines on the software development process to deal with the problem of insufficient documentation, but this necessarily increases product cost. Although documentation costs dropped significantly during last decades, it still remains a major cost factor with up to 30% of total project cost (Pressman, 2001). In research facilities where products are developed as prototypes and where organizational structures are lacking rigorous management control this becomes even more relevant. The lack holds for free/open software environments, too.

In this paper we present the Deudroid CASE tool[2] which adapts smoothly to software projects which are

[1]An exception to this is the Google Desktop Search (`http://desktop.google.com/`). But using it might conflict with a company's information security policies.

[2]Deudroid was first introduced in (Prause, 2006)

Prause C., Kuck J., Apelt S., Oppermann R. and B. Cremers A. (2007).
INTERCONNECTING DOCUMENTATION - Harnessing the Different Powers of Current Documentation Tools in Software Development.
In *Proceedings of the Ninth International Conference on Enterprise Information Systems - AIDSS*, pages 63-68
DOI: 10.5220/0002375200630068

already in progress. It will provide useful internal documentation at very low cost. The Deudroboc system cannot and is not intended to replace good documentation nor does it directly aid the process of writing documentation, but it can alleviate the problems arising from information getting lost or being undiscoverable.

Facilitating a modular structure, Deudroboc is able to automatically collect documents from various sources, to archive the documents and link them to each other, so they can be accessed more easily and more efficiently. The resulting hypermedia will be integrated into the developers' desktops. In a reference implementation we chose to make it accessible with common Internet browsers through a TCP/IP network.

In Section 2, we discuss related work for software documentation tools. We then present our concept, followed by our implementation in Section 4. In Section 5 we evaluate our approach. Finally, we conclude this paper, and discuss future work.

2 RELATED WORK

The benefits of documentation have been subject of many scientific surveys and the need for useful documentation is of paramount importance (Becker et al., 2005). Guidelines on what has to be kept in mind when writing documentation are numerous. For an example look at (Lehner, 1993). Because of this (Lethbridge et al., 2003) analyzed, which quality criteria for documentation are really important in practice.

In addition to this we want to stress the necessity of creating documentation as a parallel process while software is being developed, as this preserves most of the valuable information that should be contained in documentation (Balzert, 1988).

(Vestdam and Nørmark, 2005) have already identified the problem that often documentation is not up to date with the program's source code. With their work they focus on the growing difference between a program's documentation D_i and its source code P_i in the i-th iteration of program development, while assuming that in the beginning ($i = 0$) D_0 matches P_0 perfectly. Although they relate code to documentation texts, we think that this idea must be taken a step farther and that it is important to obtain references automatically to achieve high interconnectivity while not burdening the documentation developer.

Various classical types of tools used in software documentation have been identified and categorized into the following groups (Prause, 2006):

- *Word processors* include tools from simple text editors and Wikis to complex WYSIWYG publishing software. They do not constrain the author to what he writes, nor do they facilitate information other than that.
- *Comment extractors* transform code comments into documentation. Such tools exploit important inherent documentation (Didrich and Klein, 1996). A major advantage is the vicinity of code and documentation in one file.
- *Reporting tools* generate project status reports by incorporating up-to-date information into user defined templates.
- There are further kinds of programs and paradigms (like *Outline processors* and *literate programming*) which are not really relevant in this context.

Although the tools in one group have their respective advantages over another group, they all share a common weakness, which is their inability to combine documentation from different sources and to harness their diverse strengths by doing so. Software similar to the Deudroboc system, which serves exactly this purpose, is not known to the authors.

Another approach is to generate documentation out of the source code which guarantees an all-time correctness of the generated documentation and allows different levels of abstraction (van Deusen and Kuipers, 1999). Even though this attempt seems quite promising, it still relies on information from just a single source.

For the coordination of tasks in group work, which is the basis for collaboration, research has been done. (Malone and Crowston, 1994) found, that coordination can be seen as the process of managing dependencies between activities. (Tellioglu, 2004) specifies that dependencies happen when an activity's output is used by other activities, when multiple resources use the same resource or when multiple activities produce a common resource. She uses temporal relations between tasks to describe and manage these dependencies. In case of software documentation, dependencies arise, because multiple developers work on a single software system. The documentation that a developer produces is read and may be supplemented by other developers.

3 INTERCONNECTING DOCUMENTATION

In this section we provide an overview of the most important concepts of the Deudroboc system. The basic

idea is to automatically identify documents in various document sources[3] and then retrieve the documents from these repositories.

The process of generating interconnected documentation is subdivided into the phases:

1. analysis of documentation sources,
2. detection of dependencies,
3. generation of interconnected documentation.

Once Dendrodoc detects a new document, it is copied to an internal archive to prevent accidental loss of valuable information, when a document is removed from its respective source.

After that, the archived documents have to be categorized according to their origin and be tagged with metadata (like time stamps and keywords) for information retrieval processes in later steps. Additionally, the layout will be isolated from the document content to achieve a higher integrability with documents from other sources and to allow for a more sophisticated appliance of information retrieval processes. The layout will not be discarded completely, though, to maintain a certain level of readability.

In a second stage, interconnections between the documents will be detected. This phase makes use of explicit and implicit relations. In this notion, *explicit* means references that are already present in the documents like URLs or the `in-reply-to` header field of emails. Based on the proposition of (Tellioglu, 2004) we decide to use temporal data to identify dependencies, too. For this, we define documents as interdependent, if one document is the successor of the other doument.

An *implicit reference* denotes a reference which involves a more complex strategy in order to be gathered. This might be based on the appearing of a class's name in a document or document similarity in terms of the information retrieval vector model.

Adding more documents to an initially empty (information) space and interconnecting these documents with references to each other is essentially nothing else than creating a graph, which has documents as its nodes and references as its edges. Therefore this graph will be called the *documentation (hyper-)graph*.

In a final processing step, this graph is handed over to a visualization module which is responsible for presenting the complex graph to the user in a comfortable way.

The didactically advanced hypermedia format allows one to inspect the entire documentation with an Internet browser tool in a very user-friendly way,

[3]Such sources include, but are not limited to, email accounts, text and source files, Javadoc and CVS logs.

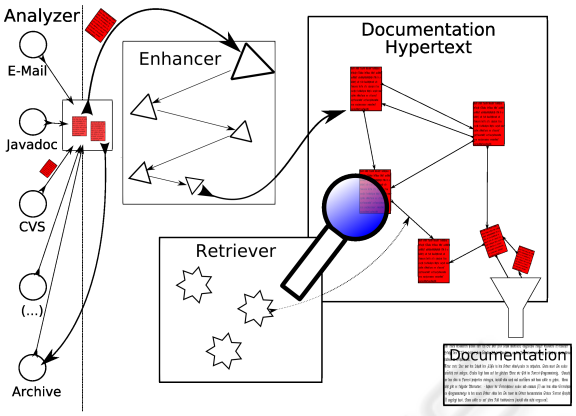

Figure 1: Documents passing through the Dendrodoc system.

based on the rich dependency links, that exist between documents with temporal, thematic or authorship based similarities.

The path of the documents through Dendrodoc from their sources to the final documentation can be seen in figure 1.

As described above, different software documentation tools can be interconnected by adequately archiving, processing and interlinking the diverse documents emerging during a software engineering process. Using an interactive data visualization, our approach assists developers in a meaningful way. Automatic support according to identify relevant documentation texts as well as browsing the documents eases handling of the documentation. Most documentation tools nowadays do not take into account different documentation techniques as base for an interconnected documentation.

Our contribution is enriching documentation and simplifying access to information — gathered out of various media types — to significantly enhance the effectiveness of software development.

4 DENDRODOC – PROTOTYPE

Dendrodoc is the platform on which the approach of archiving, interlinking and visualization of the results of multiple documentation strategies is affiliated. We will introduce the architecture of the Dendrodoc system. After that we present one of the obstacles that we encountered and provide our respective solution to it.

4.1 Design

Our prototype consists of three major subsystems, which reflect the above mentioned three phases. Each

subsystem features a plug-in API to optimally adopt Dendrodoc to project needs by allowing the development of custom add-ons for the core functionalities:

1. The **Input** subsystem which scans sources and archives for available documents
2. The **Reference generation** subsystem which links the documents to each other using information retrieval processes
3. The **Output** subsystem which prepares the generated documentation hypergraph for visualization

Dendrodoc currently uses Apache Forrest[4] on a dedicated server to provide convenient accessibility through the available intranet infrastructure. Forrest is a publishing tool which comes with an integrated web server that can convert XML Docbook files to web pages on-the-fly.

4.2 References in Dendrodoc

Generating references between documents is one of the primary tasks of the Dendrodoc system. Naturally, it is important to have a multiplicity of references between the documents to allow for a high interconnectivity of information.

The version of Dendrodoc which is used in the evaluation, is capable of generating internal references out of the following information:

- Facilitating boolean information retrieval, occurences of class names or other almost unique identifiers will be detected and linked to according documents.
- Documents with successive modify/creation time stamps will be integrated into a time path of references.
- Explicit references like `in-reply-to` headers of emails will be converted into references.
- Links between documents of different types that belong to each other logically will be established. Such connections exist between class documentations and respective source files or change logs.

Where possible, Dendrodoc integrates links into the text by turning usual words into active hyperlinks. For references where such an approach is not feasible, we group links by their respective origin in a separate *References* section at the end of a document. We also investigated weighting the links by using a scalar usefulness probability value, but discarded this idea in favor of the aforementioned grouping by reference type.

[4] `http://forrest.apache.org`

4.3 Dendrodoc Interface

The interface that is generated and delivered by Forrest is plain HTML and can thus be accessed with the user's browser of choice. Another feature that made Forrest the preferred candidate for visualizing the documentation is its search capability. Forrest comes with an integrated full-text search that allowed us to concentrate our development efforts on providing more enhanced internal references.

Besides the search function, the Dendrodoc interface provides various indexes as entry points to the documentation. These indexes list the documents from respective sources.

5 EVALUATION

In this section we present the major questions that have to be explored, how the evaluation is realized and what the outcome was.

5.1 Topics

When evaluating Dendrodoc one has to examine if and what benefits arise for developers from using it. For the time being, there is no other system which compares to Dendrodoc, so only a qualitative investigation makes sense. To measure the quality of our prototype, the evaluation concerned the following topics:

- Does Dendrodoc improve development work flows?
- Will project information be easily accessible?
- Can Dendrodoc be used in an effective and efficient way?
- Is there a positive effect on product quality?

5.2 Experimental Setup

To measure the quality of Dendrodoc, we performed an usability evaluation. Usability tests are conducted to verify that the tested software achieves user and organizational objectives. Furthermore they provide feedback for improving design (ISO 13407, 1999). From the various evaluation methods available for usability tests, we decided to use a combination of subjective methods, because these allow to infer the acceptance of a software (Opperman and Reiterer, 1997). The users' first objective was to use Dendrodoc to work through a list of tasks while verbalizing their thoughts. This so called *Thinking Aloud* technique (Ericsson and Simon, 1980) is an established and well

known technique concerning usability testing, and is especially useful in early development stages of a new software (Nielsen and Landauer, 1993). During the usability test, it may become necessary to remind the subjects to express their thoughts. After that test a semi-standardized *interview* was arranged with the subjects to capture opinions and positions. Details on the interview can be found in (Prause, 2006).

We used a documentation hypermedium based on various documents that originated from the later phases of development of the Advanced Learning Environment (ALE) platform from the WINDS[5] project. It consisted of nearly 4000 independent documents, which were linked by Deutrodoc through almost 20000 internal references.

The subjects' tasks were to get a first impression of the documentation, search for "coding rules" and specific class descriptions, find related classes and determine time, origin and kind of code changes. Then they were presented a task that required them to catch up on a certain method, get to some other class by taking source code into account and finally learn about undocumented issues with that class that had been discussed on the mailing list.

One entire usability test would take a total of 30 minutes to complete; of which twenty minutes were consumed by the usage part and ten minutes by the final interview.

The individual tests were ran with a typical number of five developers (Krahmer and Ummelen, 2004) and took place at the subjects' respective workplaces. The interviewer provided an introduction and assigned tasks one by one. Speech was recorded, so that there was no need for taking notes during the evaluation. When the respective subject had finished with his last task, the investigator immediately started the interview.

5.3 Results

The first part of the test mainly revealed usability issues related to inadequate design and layout of the documentation, which would be too detailed to present here. We restrain to a condensed presentation of the results of the questionaire:

The concept of compiling all texts into one large hypermedium was valued by all of the subjects. Four got the impression that they could get informed more easily and three could image to using the software; mainly because they thought that Deutrodoc, would make their work more efficient.

[5]Web based INtelligent Design tutoring System, http://winds.fit.fraunhofer.de/

The overall results can thus be interpreted as to support the idea of the Deutrodoc system.

6 CONCLUSION

The need for high quality software has always been an important part of software engineering. Since then it is commonly agreed on that documentation maintains a vital role in achieving product quality. This is based on the assumption that there are three primary factors in quality assurance: *process*, *technology* and *people*.

Although documentation has an impact on the process, too, it first and foremost increases motivation and quality of involved developers. But still the various positions regarding documentation may be as different as between the two extremes of ISO 9000, with its extensive documentation requirements and extreme programming, where documentation shall not hinder the development process. Especially with regard to the process, strict documentation guidelines are sometimes even considered harmful to business (Seddon, 1997).

Because of this, a lot of diverse documentation tools exist, which on the one hand, do support the development of suitable documentation and on the other hand shall reduce documentation cost. We hold the view that by no means these two aims are mutually exclusive.

In fact we think that the most important quality of documentation is its ability to inform the persons that are directly involved in the development process, without laying out any insentient documentation standards. A survey about existing documentation tools revealed the flaw that all existing tools have in common, which is their inability to make further use of existing information. Deutrodoc solves this problem by building a project memory from existing documents.

From the conducted evaluation we conclude that the usage of an advanced successor of the early Deutrodoc prototype could bring many benefits into software development processes, but some usability issues have to be addressed first.

This paper proposes the realization of a prototype for interconnecting documentation in software development. Future work on the prototype will lead in three directions. First we will investigate the impact of the adaption of social information retrieval techniques (Kirsch et al., 2006). Based on these techniques, we will measure the reputation of individual developers, concerning their documentation texts. The documentation texts of developers with a higher reputation could then obtain a higher importance.

Second, a multi-perspective documentation

(Becker et al., 2005) is a promising next step, where interconnected documentation adapts to different information needs and roles of individual users. It is also planned to enable developers to access the documentation archive, so that archived documents can be easily modified.

Third, we will examine ways of improving a user's reading experience by evaluating other visualization methods. We will design plugins for major IDEs to further reduce the sensed distance between the developers' working environment and the documention interface. Additionally we may investigate on more convenient means of presenting the complex hyperlink structure and giving navigational hints.

REFERENCES

Balzert, H. (1988). Ökonomische software-wartung durch adäquate software-konstruktion. In Wix, B. and Balzert, H., editors, *Softwarewartung*. BI Wissenschaftsverlag.

Becker, J., Janiesch, C., Delfmann, P., and Fuhr, W. (2005). Perspectives on process documentation - a case study. In *ICEIS (3)*.

Didrich, K. and Klein, T. (1996). A pragmatic approach to software documentation. Technical Report 4, Technische Universität Berlin.

Ericsson, K. A. and Simon, H. A. (1980). Verbal reports as data. In *Psychological Review, 87(3)*.

ISO 13407 (1999). Human-centered design processes for interactive systems. In *ISO/TC 159 Ergonomics, ISO International Organization for Standardization ISO 13407 (E)*.

Kirsch, S. M., Gnasa, M., and Cremers, A. B. (2006). Beyond the Web: Retrieval in Social Information Spaces. In *Proceedings of the 28th European Conference on Information Retrieval (ECIR 2006), vol. 3936 of Lecture Notes on Computer Science*, Berlin. Springer Verlag.

Krahmer, E. and Ummelen, N. (2004). Thinking about thinking aloud: A comparison of two verbal protocols for usability testing. In *IEEE Transactions on Professional Communication, 47(2)*.

Lehner, F. (1993). Quality control in software documentation based on measurement of text comprehension and text comprehensibility. *Inf. Process. Manage.*, 29(5).

Lethbridge, T. C., Singer, J., and Forward, A. (2003). How software engineers use documentation: The state of the practice. *IEEE Software*, 20(6).

Malone, T. W. and Crowston, K. (1994). The interdisciplinary study of coordination. *ACM Comput. Surv.*, 26(1).

Nielsen, J. and Landauer, T. K. (1993). A mathematical model of the finding of usability problems. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, New York, NY, USA. ACM Press.

Opperman, R. and Reiterer, H. (1997). Software evaluation using the 9241 evaluator. *Behaviour and Information Technology*, 16(4/5):232–245.

Prause, C. (2006). Design und evaluation von dokumentations- und qualitätssicherungsmethoden am beispiel der fit-lernplattform. Master's thesis, Rheinische Friedrich-Wilhelms-Universität, Bonn.

Pressman, R. S. (2001). *Software Engineering: A Practitioner's Approach*. McGraw-Hill.

Seddon, J. (1997). *In Persuit of Quality: The Case Against ISO 9000*. Oak Tree Press.

Tellioglu, H. (2004). CoMex - a mechanism for coordination of task execution in group work. In Cordeiro, J. and Filipe, J., editors, *Computer Supported Activity Coordination, In conjunction with ICEIS 2004*. INSTICC Press.

van Deusen, A. and Kuipers, T. (1999). Building documentation generators. In *Proceedings IEEE International Conference on Software Maintenance 1999 (ICSM'99)*. IEEE.

Vestdam, T. and Nørmark, K. (2005). Toward documentation of program evolution. In *Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM'05)*. IEEE.