A View-Based Approach for Improving Software Documentation Practices

Joachim Bayer and Dirk Muthig
Fraunhofer Institute for Experimental Software Engineering (IESE)
Fraunhofer-Platz 1, D-67663 Kaiserslautern, Germany
{joachim.bayer, dirk.muthig}@iese.fraunhofer.de

Abstract

Documentation is an integral part of a software system. It contains the information that is necessary to effectively and successfully develop, use, and maintain a system. In practice, however, the creation of appropriate documentation is largely neglected.

This paper investigates the reasons for this neglect, presents view-based software documentation, our approach to improve the current situation, and reports on empirical evidence in support of the presented approach. Because the quality of documentation depends on its usage, view-based software documentation exploits existing software modeling techniques to provide all users of documentation with the documentation they require for performing their tasks.

View-based software documentation has been empirically validated in a series of experiments and case studies that showed that the approach improves the completeness, correctness, and usefulness of produced and maintained documentation. One of the experiments is described here in more detail.

1. Introduction

Software engineering is the discipline that provides principles, techniques, methods, and tools to systematically develop software systems. The development of a software system encompasses the development of the executable program and the creation of appropriate documentation. In practice, however, the creation of appropriate documentation of software is largely neglected.

Different stakeholders have different tasks related to a software system that should be supported by appropriate documentation. Software architects, project managers, or programmers, for example, contribute to different aspects of software development. A second group of stakeholders are testers or people that decide whether the software is bought by some organization; this group of stakeholders assesses or evaluates some aspects of the software. A third group of stakeholders is using the software. Other stakeholders are the people that create or maintain the documentation. All stakeholders performing these tasks need diverse knowledge about the software system to perform their tasks. The knowledge the stakeholders require can be provided to them by a documentation that contains the information they need to perform the respective tasks. The type and presentation of the information is, therefore, dependent on the performed task and the stakeholders.

In contrast to the need for documentation stands the current, widespread practice that software development organizations do often document their products after the implementation is complete. Often documentation is done the minimal way producing only contractually guaranteed documents; usually these are the user documentation and whatever the client requires.

Being asked about the reasons for them to document the absolute minimum or the reasons they start documenting afterwards, software developers raise similar issues with documenting the software they are developing: in many organizations there is simply no established process that integrates the documentation into development making it an integral part of the final product. If, in addition to that, an organization does not consider documentation as important as the 'real' product - the source code - the motivation for software developers to create and maintain documentation becomes very low.

Even if there is an established process for documentation, often documents are produced according to given rules without rationales and it is, therefore, not clear to the producers of the documentation what is being done with the documents they produce and for whom they are producing them. The result is that many people while documenting have the subjective impression of creating too much unnecessary documentation that is produced only for the sake of documenting. The colloquial term for that kind of documentation is shelfware. Naturally, producing shelfware is not a motivating and satisfying task, which decreases the quality of the results even more. The basic underlying problem is that neither the purpose of the documentation nor the people that consume the documentation are known to the producers of the documentation.

To benefit from documentation in cases in which it is beneficial and necessary without being trapped in producing and using unnecessary, unreadable, and non-beneficial documents, documentation should be of a high quality. What a documentation of high quality is depends on the usage of the documentation. The different users of documentation with their expectations and requirements must, therefore, all be taken into account when the quality of software documentation should be improved.

In this paper, we present and validate view-based software documentation, our approach based on the software engineering principle separation of concerns to develop customized documentation [7]. This paper is organized as follows. Section 2 presents the current state-of-the-art and state-of-the practice for documenting software systems. Section 3 then presents the view-based approach to documenting software systems. Section 4 contains the description of the empirical validation we conducted to validate the approach. Section 5 finally summarizes and concludes the paper.

2. Context

A recent study revealed that natural language is the predominant notation in software documentation, that most of the participating companies have problems in keeping the different documents produced consistent, and that the documents are not correct any more by the end of a project [19]. The findings of this study is consistent with results of earlier, similar studies, for example [11] [16].

There are a number of different approaches that are proposed to document software. At one extreme, there are the agile software development approaches that assert that source code alone is sufficient as documentation, for example eXtreme Programming (XP) [8]. The benefit of such an approach is that the documentation (i.e., the source code) is always consistent with the software system. Some approaches augment agile approaches with means for documentation, for example the agile modeling approach [1]. The agile modeling approach classifies models into models that are developed to support the development of the software and models that are developed to document the software system. For the documentation, models, techniques are given that realize an agile approach for the documentation of software systems.

At the other extreme of the range there are approaches that define a complete documentation schema to be used. These strict documentation approaches propose a structure that the producer of the documentation has to fill to document a software system. Examples for strict documentation approaches include standards, like for the user documentation [15] or for the requirements specification [14]. Some software engineering books also recommend structures for documenting requirements [9] [23] or reuse documentation [21].

Books about the documentation of software are rare. There are, however, some books that primarily address software documentation, for example [3] [20]. Both stress that documentation must be written for the consumers of the documentation and provide guidelines or patterns that support achieving this goal. Both, however, do not provide any systematic approach to elicit the informational needs of the documentation consumers.

In this paper, we present a systematic approach that realizes the software engineering principle separation of concerns to enable the definition of documentation schemata that are tailored to the respective project context.

Separation of concerns is a software engineering principle that promotes the isolated descriptions of all important aspects of a software system and that, therefore, will be used as a basis for developing a systematic approach to improve the current state-of-the-practice in documenting software systems.

A number of approaches have been proposed to realize an explicit separation of concerns. The notion “aspect-oriented” is generally used to characterize these approaches. The notion was introduced in the aspect-oriented programming (AOP) approach [17]. Other approaches for realizing separation of concerns at the implementation level are subject-oriented programming (SOP) [12] and multi-dimensional separation of concerns [22]. There are also aspect-oriented software development approaches that address earlier life-cycle stages. Examples are aspect-oriented requirements engineering [10] or architectural views [13] [18].

3. Method for View-Based Documentation

3.1. Concepts

Separation of concerns is the attempt to identify, describe, and handle the different concerns stakeholders have on a software system separately and in isolation. A concern is any matter of vested interest that a stakeholder has in a software system.

There are three major challenges that have to be addressed to achieve separation of concerns, the elicitation, the explicit separation, and the composition of concerns. Once these three challenges are solved, a view-based documentation of software is possible. A view-based software documentation supports different viewpoints on the software system and enables the explicit and simultaneous modeling of all of those viewpoints as views in the documentation.

A viewpoint is a perspective taken by a stakeholder towards a software system. A viewpoint specifies a metamodel that expresses the type of entities and relationships that are of interest from the perspective the viewpoint defines. A view instantiates a viewpoint and conforms to the viewpoint's rules. A view uses the types that are defined in the corresponding viewpoint, and describes the software system using concrete entities that are instantiations of the types given in the metamodel of the respective viewpoint — similar to the distinction between classes and objects in object-oriented approaches.

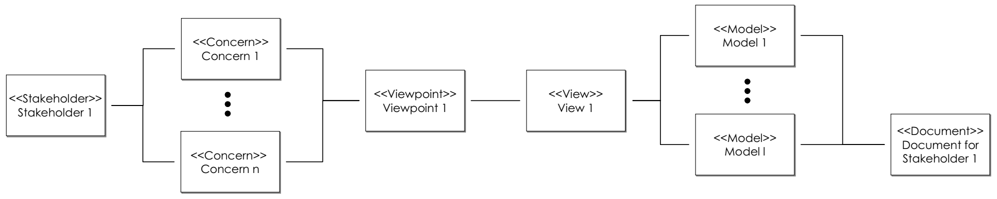
Figure 1 shows the situation that view-based documentation aims at. The concerns of the different stakeholders are elicited, and viewpoints are created that reflect for each stakeholder their perspective on the software system. Based on the viewpoints, models are created and used as document elements in a document that exactly provides the information the respective stakeholder needs.

3.2. Principal Approach

Separation of concerns is applied in view-based documentation to the documentation of software systems. The reason for this approach is that the quality of software documentation is dependent on the use of the documentation. The consumers of a documentation and their subjective assessment of its quality are, therefore, the most important input to understand and improve a documentation's quality. Consequently, a documentation's stakeholders and their concerns, that is, the information they want to find in the documentation, are the optimal starting point for defining a documentation schema. Such a documentation schema defines what information a documentation captures, how this information is presented to the readers of the documentation, and what documents the documentation consists of. The result is one documentation per stakeholder that contains exactly the information the stakeholder requires for working with the documentation (compare Figure 1). It is possible that there are certain concerns, and, therefore, also parts of viewpoints that overlap. The viewpoints are, however, treated separately to determine the ideal viewpoint for each stakeholder. This viewpoint can then be taken as a basis for developing a documentation schema that yields a documentation that is optimal for the respective stakeholder.

In practice, however, an approach that provides one customized document for each stakeholder separately is hardly feasible. The effort for planning, creating, and maintaining such a documentation has no reasonable relation to the reduction of effort that can be reached for consuming the resulting documents. The quality of documentations for the different consumers can, however, be improved without creating a documentation for each of them. The approach that is introduced here is based on the fact that there is a substantial overlap among the different viewpoints. A viewpoint determines the information that is contained in the models that are used to represent certain concerns. Overlapping viewpoints consequently share models and their elements. Models can, therefore, be used in different documentations and so a pool of models can be created from which models can be taken to develop different documentations.

Another aspect of the overlap of viewpoints is the fact that there is often a basic set of information that is required by a large number of documentation



<<Stakeholder>> Stakeholder 1
<<Concern>> Concern 1
<<Concern>> Concern n
<<Viewpoint>> Viewpoint 1
<<View>> View 1
<<Model>> Model 1
<<Model>> Model l
<<Document>> Document for Stakeholder 1

Figure 1. View-Based Documentation

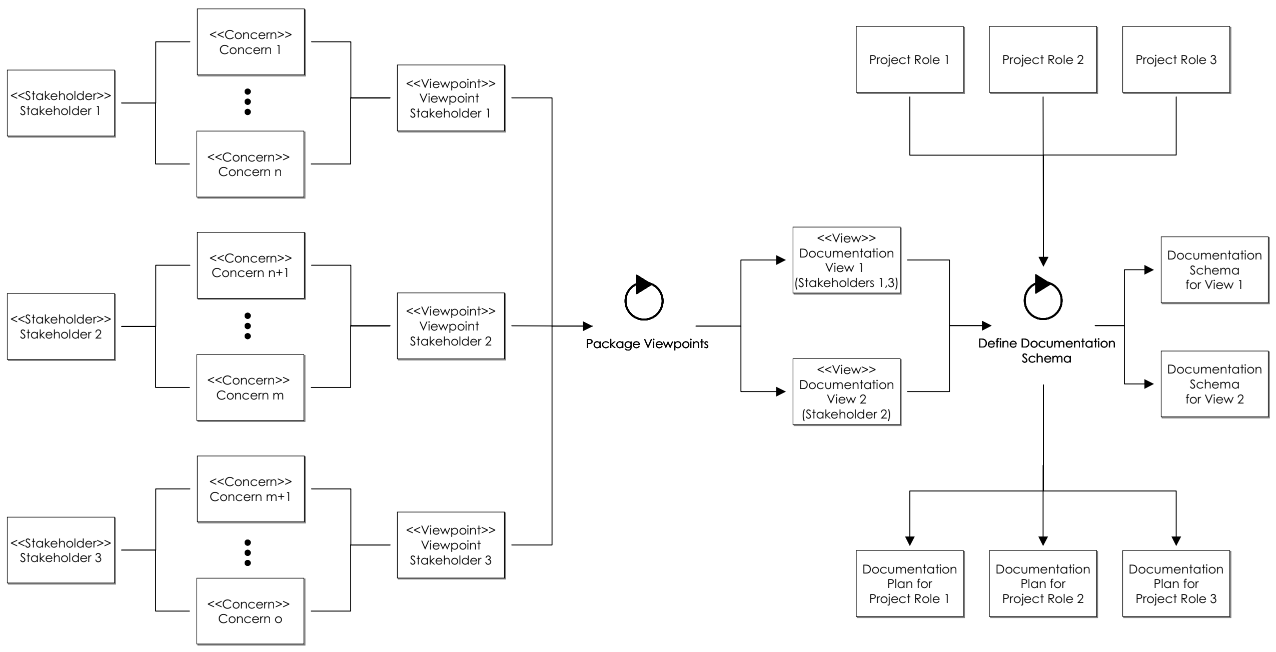

<<Stakeholder>> Stakeholder 1
<<Concern>> Concern 1
<<Concern>> Concern n
<<Viewpoint>> Viewpoint Stakeholder 1
<<Stakeholder>> Stakeholder 2
<<Concern>> Concern n+1
<<Concern>> Concern m
<<Viewpoint>> Viewpoint Stakeholder 2
<<Stakeholder>> Stakeholder 3
<<Concern>> Concern m+1
<<Concern>> Concern o
<<Viewpoint>> Viewpoint Stakeholder 3
Package Viewpoints
<<View>> Documentation View 1 (Stakeholders 1,3)
<<View>> Documentation View 2 (Stakeholder 2)
Project Role 1
Project Role 2
Project Role 3
Define Documentation Schema
Documentation Schema for View 1
Documentation Schema for View 2
Documentation Plan for Project Role 1
Documentation Plan for Project Role 2
Documentation Plan for Project Role 3

Figure 2. Approach to View-Based Documentation

consumers. For software documentations, this is usually the information that covers the functionality and features of the respective software system and how these are realized in the software system. This is the information that is typically covered by the documentation schemata provided by general-purpose approaches to the development of software systems. Even though not every stakeholder requires all information in such basic viewpoints, they can be used to as a skeleton to which stakeholder specific aspects can be added to yield customized documentations.

Figure 2 depicts the approach to view-based software documentation. The viewpoints for the different stakeholders are still elicited. However, there is an packaging step that analyzes the viewpoints and consolidates the elicited viewpoints into an optimal set of views. The optimization criteria are the number of models that are to be produced for each stakeholder and the number of models produced in total. To optimize the presented information, both numbers should be as small as possible under the constraint that a possibly large number of the constraints are covered. As shown in figure 2, the result is a number of views that encompass information for the different stakeholders that will be present in the different documents for the different stakeholders.

View-based software documentation can be developed based on the resulting optimized views. The views are taken as basis to create a documentation schema from which an actual view-based documentation can be derived. In this step, also the creation of the documentation is planned. The different project roles, that is, the staff working on a project and their role in the project, are taken as input to define the documentation plans for the different roles. A documentation plan determines the document elements and documents the different people working in a project must develop.

The documentation of software should not be addressed in isolation for each project a software developing organization performs. On the contrary, it is beneficial to view the documentation of the software an organization produces in an organization-wide manner and to learn from past experience in documenting software in earlier projects. This approach is based on a framework for integrated product and process engineering that was developed within the TAME (Tailoring a Measurement Environment) project [4]. The principal goal of the framework is to record and package experience in an experience database and to use the information captured in the experience database to improve the software development practices in an organization.

The information that is recorded for the different projects is then packaged to yield an experience database that supports the preparation and execution of new projects. For documentation, the experience database contains a organization-wide documentation schema as product model along with information on how it was adapted in different projects. Information on how the different documentation schemata reflect project characteristics and what the experience was with that approach are also captured. Additionally, a quality model for documentation is contained in the experience base that enables the evaluation of documentation schemata and documentations.

The information and experience stored in the experience base can be used to support new projects in the organization. The organization-wide documentation schema can be used as a starting point for preparing a documentation schema for a new project. Characteristics of the new project are compared to the characteristics of past projects and, taking into account the experience gained in the past projects, the organization-wide documentation schema is modified to yield a new documentation schema.

3.3. Approach in Detail

The preceding section presented a view on software documentation as a organization-wide resource that can be continuously improved. Figure 3 shows the typical life-cycle of a software documentation that consists of three steps as an IDEFØ diagram. First, the documentation schema is determined that defines the documents to be produced, as well as their content in terms of the information being contained and the notations used to represent that information. The documentation plan is also created that contains the schedule for creating the different parts of the documentation.

Once the documentation schema and plan are defined, the documentation is built. The different documents are produced and delivered to the documentation consumers that use it to work with the software system being documented.

The three identified phases are presented in more detail in the following three sections.

Documentation Definition

The goal of the documentation schema definition process is to set up a documentation schema that covers as much of the relevant information on the software system as possible and to present it to its consumers in a way that suites them optimally. Additionally, a documentation plan is developed that schedules and plan the creation of the documentation.

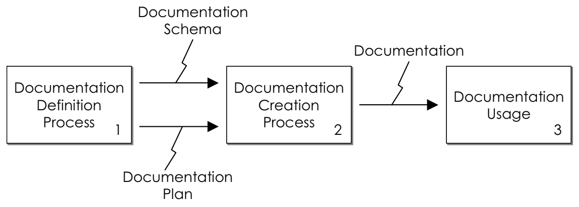


Figure 3. Life-cycle of a Documentation

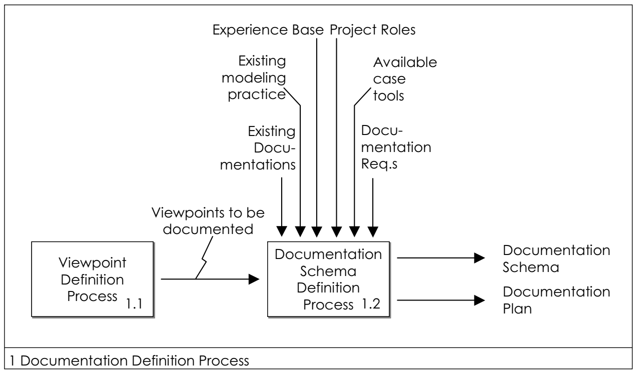

Experience Base
Project Roles
Existing modeling practice
Available case tools
Existing Docu-mentations
Docu-mentation Req.s
Viewpoints to be documented
Viewpoint Definition Process 1.1
Documentation Schema Definition Process 1.2
Documentation Schema
Documentation Plan
1 Documentation Definition Process

Figure 4. Documentation Definition Process

The documentation definition process is shown in Figure 4. It consists of the two activities, viewpoint definition and documentation schema definition.

In the viewpoint definition, first, the stakeholders that are documentation consumers and their concerns are elicited. The goal of the viewpoint definition is to elicit and document the viewpoints of all stakeholders (Step 1.1 in Figure 4) and to plan the creation of the actual documentation. The result of the viewpoint definition activity is a definition of the viewpoints that are to be covered by the documentations being produced and that, therefore, are used as a basis to develop the documentation schema.

The goal of the actual documentation schema definition is to create a documentation schema that covers the defined viewpoints (Step 1.2 in Figure 4). The obvious goal is to cover all documentation relevant concerns in exactly the way the stakeholders want it. The resulting documentation set that provides for all stakeholders exactly what they want is usually too expensive to create and maintain. Then, the trade-off between concern coverage and cost must be resolved. Viewpoints can, for example, be combined into composed views to cover concerns from different stakeholders' viewpoints. These measures result in documentations for more than one stakeholder. As mentioned in the introduction to this chapter, it is not realistic to expect cost effective complete concern coverage. The set of views in the documentation is optimal if the number of views is as small as possible, yet covering all relevant concerns. In the best case, the set of views directly corresponds to the viewpoints elicited in the viewpoints elicitation step. Often a number of viewpoints can not be covered in a documentation in a cost-effective way and, therefore, must be merged with others.

If no experience is available from an experience base, existing modeling practices, existing documentations, and CASE (Computer-Aided Software Engineering) tools are investigated to

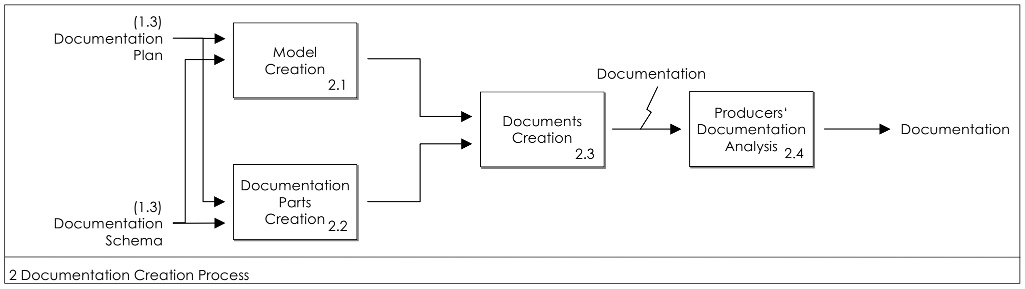

(1.3) Documentation Plan
(1.3) Documentation Schema
Model Creation 2.1
Documentation Parts Creation 2.2
Documents Creation 2.3
Documentation
Producers' Documentation Analysis 2.4
Documentation
2 Documentation Creation Process

Figure 5. Documentation Creation Process

understand to what extent the existing modeling practices cover the required documentation viewpoints. Additionally, specific requirements on the resulting documentations are considered when developing the documentation schema. If there is an experience base available, it is investigated whether the current concerns are already covered there. The respective the definition of views and documentation schemas ca then be reused. The changes and adaptation that have been made, as well as the rationales for these, are recorded in the project database. The result of the documentation definition process is a documentation schema that defines the structures for the different documents the different documentations to be produced consist of and that relates the viewpoints to these documentation structures.

The creation of the documentation is planned as well in the documentation schema definition process. The roles participating in a project are used to plan and schedule the creation of the different elements of the documentation. This is captured in the documentation plan.

Documentation Creation

The documentation creation process is shown in Figure 5. It consists of four activities, view creation, documentation parts creation, documents creation, and producers' documentation analysis.

The first activity towards creating the documentations defined in the preparation phase is to create the models for the views to be documented based on the documentation plan (Step 2.1 in Figure 5). As defined above, a view is a number of models that together represent a viewpoint. The models in a view collaborate and jointly represent the perspective taken on a software system that is defined in a viewpoint. The metamodel given in a viewpoint selects and relates the information present in the respective models. The result of the model creation activity is a set of models that can be used in the planned documentations. Therefore, the models must be organized and identifiable.

A documentation mainly consists of models. There are, however, also other parts of the documentation that need to be present, for example, explanatory text, change histories, author lists, or contact information for a hotline to be addressed in case of problems. All of these are written in the course of the documentation part creation (Step 2.2 in Figure 5). The goal of the documentation part creation activity is to write the texts that will appear in the documentations. This includes all sorts of text from headline texts to descriptions of the methods used to develop the software being documented.

The documents creation activity concludes the documentation creation (Step 2.3 in Figure 5). In this step, the views and the document parts created are integrated in the documentation schema. Since the documentation schema identifies the places where the different models and document parts are located in the different documentations, the integration can be easily automated using tool support. This enables an efficient building and maintenance of documentations. It also supports the usage of the same models and documentation parts in documentations and the presentation of documentations in different output formats.

After the different producers have created their share of the documentation, it can be evaluated from the producers' perspective (Step 2.4 in Figure 5). The evaluation assesses whether a producer created all required elements of the documentation and tries to understand why certain elements have not been produced (if applicable). Other information captured is the correctness and consistency of the documentation schema, as well as the subjective impression of the producers with respect to the usefulness of the documentation schema. To analyze the effort required for documentation production, the effort that was spent for documentation within development and in addition to development are collected.

Documentation Usage

The customized documentations are provided to the respective stakeholders. It is essential for the success of a view-based documentation approach to get feedback on the experience documentation consumers have using the documentation. The feedback is an important source for continuously improving the documentation and its documentation schema after it is in use.

The continuous improvement of documentation in an organization requires the assurance of the quality of documentation. From a specific user's perspective, the completeness of documentation with respect to this user's tasks is an important indicator for the quality. Additionally, the correctness and consistency of the documentation, the effort required for using the documentation, as well as the subjective assessment of the usability of the documentation are important measures for analyzing a documentation.

4. Validation

The approach for view-based documentation of software was validated in a number of experiments and industrial case studies that all investigated different aspects of the approach. The experiments revealed an increase of the average completeness of produced documentations of up to 37% by using a proper documentation schema. The errors in produced and maintained documentations could be decreased by factor 3 in the best case. The subjective usability could also be increased by a proper documentation schema. In the following, one of the experiments is described in more detail.

The experiment described collected data to investigate four aspects of the quality of the produced and maintained documentations. Of these four aspects, completeness, correctness, usefulness, and economy, only one, completeness, is described in more detail. The results for the three other aspects are briefly summarized at the end of the experiment description.

4.1. The Library System Experiment

Experiment Design

The library system experiment aimed at eliciting the impact the documentation schema has on the quality of produced and maintained documentations. Therefore, experienced subjects (i.e., graduate computer science students and software engineering researchers) were asked to create and maintain the documentation of a software component. They were randomly put in one of the groups (G1 and G2) using different documentation schemata (G1 used DS1 and G2 used DS2).

Both documentation schemata proposed the same set of models to be created to document software components and differed in the presentation of the models only. The experiment was performed on a pen and paper basis. The experiment material briefly described the models to be used for documentation on top of a page followed by enough empty space to draw some models. The models that were proposed to be used were the same for both documentation schemata, namely UML class diagrams, UML object diagrams, textual descriptions of the operations provided by the component, UML state chart diagrams, UML activity diagrams, and UML collaboration diagrams. Hence, the information to be documented was the same for both groups.

The documentation schema DS2 is a documentation schema that simply listed the models to be produced with a short motivation of each of the models. Documentation schema DS1 is based on the KobrA component model [2] and also contained brief guidelines that explain the viewpoints that are documented, the models used, and how the different viewpoints are related. Additionally, the subjects were introduced to the documentation schema in a 10 minute presentation. The component that was used in the experiment was taken from a case study implementation of a library system. This case study can be found in [5]. The material also contained an implementation of the LoanManager component as an Enterprise Java Bean (EJB) that provides seven public operations and is about 250 LOC long.

The experiment was done in two passes for both groups, dealing with the creation and the maintenance of documentation, respectively. The task in pass 1 was to document the LoanManager according to the respective documentation schema using pen and paper.

In the second pass, the same subjects were asked to maintain the documentation of the same component to reflect changes that had been made to the source code. The task was to mark the required changes in the printed documentation of the LoanManager to reflect the changes in the source code. The goal was that the documentation completely and correctly documents the printed source code again after the documentation maintenance.

Additional materials used were the pre-briefing questionnaire and the debriefing questionnaire, which the subjects were asked to answer. The pre-briefing questionnaire captured the subjects' experience and on their motivation for performing well in the experiment. The debriefing questionnaire captured the subjects' opinions on the documentation schema they used, as well as their opinion on their performance during the experiment and on the experiment itself.

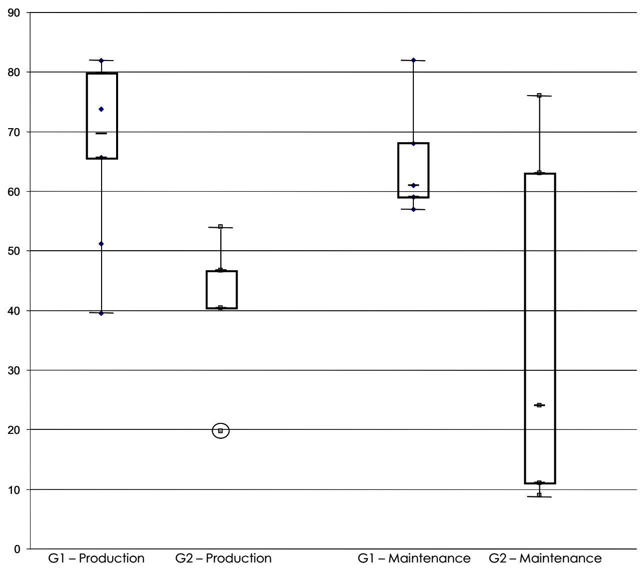


Figure 7. Completeness Results

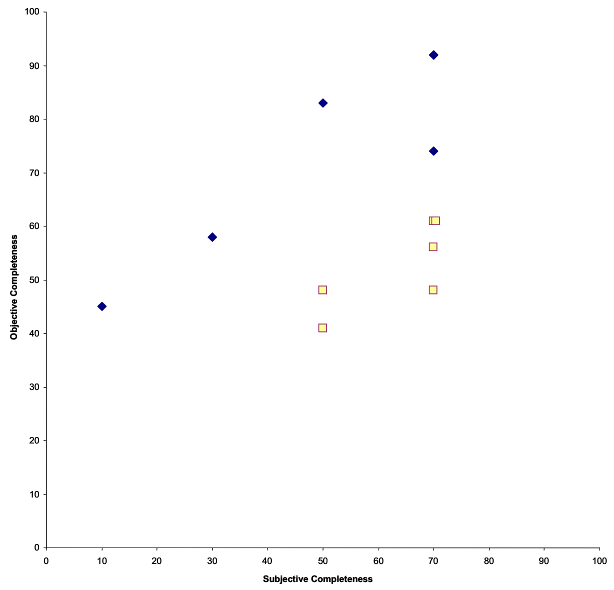


Figure 8. Subjective vs. Objective Completeness

Experiment Results

For the production task, a proposed or sample solution was created that determined the required models, as well as the required model elements in these models. This solution was the same for both documentation schemata. The solutions provided were analyzed for the models they contained, as well as for the model elements in these models. The result of the analysis is a percentage that expresses to what degree the different subjects produced a complete documentation. The analysis results can be compared for the two documentation schemata since the expected models were the same.

For the maintenance task, the completeness of changes was analyzed. Since a complete and correct documentation was given to the subjects, the required changes can be counted. The changes the subjects made by marking the appropriate places in the provided documentation were then compared to the required changes resulting in a percentage capturing the completeness of the maintenance.

The experiment was conducted with 12 experienced subjects randomly distributed in the two groups. Due to the small size of the groups the statistical power of the experiment results is low. Therefore, a qualitative, rather than a quantitative, statistically funded, analysis of the experiment results was indicated. As a consequence, the conclusions are drawn in an argumentative manner in the following to evaluate the hypotheses of the experiment.

Figure 7 shows the completeness results achieved for the two tasks as box plots. The figure shows that the completeness was higher for the documentation schema DS1 for both tasks. This suggests that the documentation schema can have a beneficial effect on the completeness of produced and maintained documentations. Another very interesting result of the experiment was revealed by the comparison of the subjective and the objective completeness. The subjective completeness was elicited using a question in the de-briefing questionnaire. The subjects were asked to estimate the completeness of the documentation they produced for the production task or the completeness of the changes they applied for the maintenance task, respectively. The completeness was elicited as a percentage on a nominal scale with 20% steps.

Figure 8 compares the subjective and the objective completeness for the production task. On the horizontal axis, the subjective completeness values are given by using the center of the respective range as value. On the vertical axis, the completeness values that have been elicited as described above are given. Points on a diagonal starting at (0%, 0%) and ending at (100%, 100%) would mean that the subjects correctly estimated the completeness of the result of their respective task. Points below that diagonal mean that the subjects overestimate the completeness of their work, that is, the subjective completeness value is higher than the objective completeness value. Points above mean that the subjects underestimate the completeness of their work, since the objective value is higher than the subjective value.

It turned out that the users of the unstructured documentation schema DS2 (depicted as squares) tended to overestimate the completeness of their work and the users of the documentation schema DS1 (depicted as diamonds) tended to underestimate the completeness of their work. A reason for this phenomenon might be that the unstructured documentation schema DS2 does not provide as much guidance that helps its users to know when they have documented or maintained a component completely.

For the documentation schema DS1, the situation is slightly different. As described above, the subjects were introduced to the documentation schema in a short presentation. The KobrA method, on which the documentation schema based, was new to them. They were familiar with the notation used but the organization, as well as the relationships among the different models that were presented to them was new to the subjects. A task setup of that kind, where a subject is introduced to a new technique and is asked to apply the technique right away, tends to overwhelm subjects easily. At least, they have the subjective impression that they can not fulfil the large number of requirements posed by that new technique on them. And even though the results are good (as the analysis presented above shows), the subjects tend to underestimate their contribution.

This phenomenon showed up for the maintenance task as well. In the second pass, the users of documentation schema DS1 estimated the completeness more correctly. This result suggest that the subjects, being familiar with the documentation schema have been able to estimate their work more correctly.

The results of the different analyses presented in this section suggest that the documentation schema can have a beneficial on the completeness results when producing or maintaining documentations. The users of the view-based documentation schema DS1 that made explicit the purpose of the different models within the documentation and that also made explicit the relationships among the different model elements achieved generally better results with respect to completeness.

The analysis of economy did not lead to clear results due to the experiment setup. The data collected during the experiment is not significant enough to justify to draw any statistically funded conclusion with respect to economy from the experiment.

The analysis of the correctness and usefulness of the produced and maintained documentations also suggest that a view-based documentation schema has a positive influence on these two aspects of documentation quality.

4.2. Other Empirical Validations

Two additional experiments were conducted, one with a larger number of inexperienced subjects to elicit the impact of experience on the results. In the other experiment, the subjects produced documentations using the two documentation schemata to directly compare the usefulness. A number of insights could be drawn from these experiments:

- Experienced subjects have better completeness and correctness results
- Missing experience can be compensated by a good, supporting documentation schema
- Experience does not compensate a low quality documentation schema
- The positive effect of the experience on the completeness of the produced is higher for the more unstructured documentation schema DS2.
- Inexperienced subjects tend to underestimate their results in producing a documentation and overestimate their results in maintaining a documentation.
- The subjective usefulness of the documentation schema DS1 is higher than of DS2.

More details on the experiments and their results can be found in [7].

View-based software documentation was also validated in a number of industrial case studies. One case study was performed in the context of an industrial project that aimed at modeling and implementing reusable software components. The case study showed that an adaptation of the documentation schema can be easily achieved and improves the quality of the respective software documentations. In another case study, view-based software documentation was used to determine the views to document a product line architecture and to provide traceability among the different views [6].

5. Conclusions

This paper presented an organization-wide approach to improve the current practice of software documentation. The work on this approach revealed that the quality of documentation is directly related to the use of the documentation. This led to the development of a quality model (not presented in this paper due to space limitations) that enables the task-related and subjective assessment of the quality of a software documentation.

The empirical validation based on the quality model showed that the quality can be improved when the documentation schema considers the different uses of the documentation. The integration in an experience-base supported environment enables the reuse of documentation practices and quick reaction on changing projects characteristics by adapting the practices in a systematically supported way. The approach proposes information models as a basis to develop documentation schemata. This should be properly supported by tools that help manage the information models.

Another aspect that should be supported by tools is the generation of documentation from a documentation schema. Another thread that should be followed is the creation of an experience base for documentation practices that captures the documentation practices proposed by different software development methods.

The conducted experiments did not cover the gain that is supposedly achieved over time by effort reduction of using high quality documentations over the effort needed to create them.

We have successfully applied the view-based approach to documenting software systems presented in this paper in a number of industrial projects and further improve it based on the experiences gained.

References

[1] S. W. Ambler. agile modeling. John Wiley & Sons, 2002.

[2] C. Atkinson, J. Bayer, C. Bunse, E. Kamsties, O. Laitenberger, R. Laqua, D. Muthig, B. Paech, J. Wüst, and J. Zettel. Component-based Product Line Engineering with UML. Addison-Wesley, 2001.

[3] T. T. Barker. Writig Software Documentation - A Task-Oriented Aproach. Allyn and Bacon, 1998.

[4] V. R. Basili and H. D. Rombach. The TAME Project: Towards Improvement-Oriented Software Environments. In IEEE Transactions on Software Engineering, pp. 758-773, vol. 16, no. 6, June 1988.

[5] J. Bayer, D. Muthig, and B. Goepfert. The Library System Product Line - A KobrA Case Study. Technical Report 024/01.E Fraunhofer IESE, 2001.

[6] J. Bayer and T. Widen. Introducing Traceability to Product Lines. In Proceedings of the 4th International Workshop on Software Product-Family Engineering (PFE-4), 2001.

[7] J. Bayer. View-Based Software Documentation. PhD Thesis, Fraunhofer IRB Verlag, 2004.

[8] K. Beck. extreme Programming explained: embrace change. Addison-Wesley, 1999.

[9] A. M. Davis. Software Requirements: Objects, Functions, and States. Prentice Hall, 1993.

[10] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke. Viewpoints: A Framework for Integrating Multiple Perspectives in System Development. In International Journal of Software Engineering and Knowledge Engineering, Vol. 2, No. 1, pp. 31-57, 1992.

[11] L. Groves, R. Nickson, G. Reeve, S. Reeve, and M. Utting. A Survey of Software Development Practices in the New Zealand Software Industry. In Australian Software Engineering Conference, pp. 189-201, 2000.

[12] W. Harrison and H. Ossher. Subject-Oriented Programming (A Critique of Pure Objects). In Proceedings of the 8th Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'93), 1993.

[13] C. Hofmeister, R. Nord, and D. Soni. Applied Software Architecture. Addison-Wesley, 1999.

[14] IEEE Recommended Practice for Software Requirements Specification, IEEE Standard 830, 1998.

[15] IEEE Standard for Software User Documentation, IEEE Standard 1063, 1987.

[16] I. John and D. Muthig. The State of the Practice of Systematic Software Development / Product Line Development in Germany, ViSEK Report 010/E 2003.

[17] G. Kiczales, J. Lamping, A. Medhekar, C. Medea, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP'97), June 1997.

[18] P. Kruchten. The 4+1 View Model of Architecture. In IEEE Software, pp. 42-50, November 1995.

[19] T. Olsson and P. Runeson. Document Use in Software Development: A Qualitative Survey. In Software Engineering Research and Practice in Sweden (SERPS'02), 2002.

[20] A. Rüping. Agile Documentation - A Pattern Guide to Producing Lightweight Documents for Software Projects. John Wiley & Sons, 2003.

[21] J. Sametinger. Software Engineering with Resuable Components. Springer, 1997.

[22] P. Tarr, H. Ossher, W. Harrison, and S. M. Sutton, N Degrees of Separation: Multi-Dimensional Separation of Concerns. In Proceedings of ICSE'99, May 1999.

[23] H. van Vliet. Software Engineering. Principles and Practice, 2nd ed., John Wiley & Sons, 2000.