4everedit – Team-Based Process Documentation Management*

Michael Meisinger
Institut für Informatik
Technische Universität
München
Boltzmannstr. 3
D-85748 Garching
meisinge@in.tum.de

Andreas Rausch
Fachbereich Informatik
Technische Universität
Kaiserslautern
Gottlieb-Daimler-Straße
D-67653 Kaiserslautern
rausch@informatik.uni-kl.de

Marc Sihling
4Soft GmbH
Mittererstraße 3
D-80336 München
sihling@4soft.de

Abstract

The introduction and successful application of engineering processes demands a modern infrastructure which supports efficient and convenient management of the various work products, their life-cycle, and their relations to each other. Mostly, work products are realized in form of electronic documents which can easily be edited, stored, copied and exchanged. However, most infrastructures are fundamentally limited by the fact that the information contained in many documents is not accessible because its semantics is unknown.

For example, software engineers often document the design of a software architecture in a rather unstructured way and hence cannot benefit from automatic evaluations such as consistency checks regarding the document itself (is it complete?) or its dependencies to other documents (is there a test specification for each component?).

Because most documents are inherently structured as a result of applying certain methods (such as CRC-Cards for software requirements engineering) and because techniques are available to model and handle structured documents (such as XML), combining both successfully seems to be a small gap to bridge.

In this paper we present the essential concepts to support team-based editing of structured text-based engineering documents. We report on our experiences from the application of the infrastructure "4everedit" in a large-scale industrial process engineering project.

Keywords: document management, process documentation, team-based editing, consistency checking, document generation

1. Introduction

During development projects in engineering disciplines such as construction, mechanical and software engineering, many different types of electronic work results are created, edited and post-processed by hand as well as by tools. We propose to distinguish these products into three categories:

- *Integrated models*, which generally contain complex models and diagrams in binary format. Examples are software architecture and design models in UML and CAD drawings. Engineers, for instance, use CASE, CAD and CAE tools to edit them.
- *Tool processed documents*, which are in general plain text documents that conform to a formal syntax that can be parsed by tools. Such files are read or generated by transformation and interpretation tools, such as compilers and interpreters. Typical examples are source code or configuration files.
- *Structured text-based documents* contain textual information that follows a predefined – often hierarchical – structure. Such documents are frequently used for specification, documentation and management purposes. Requirements specifications, software architecture documents, process documentations and project status reports fall into this category. Often, proprietary tools with binary file formats such as Microsoft Word™ are used to edit and store structured text-based documents.

* This work was partially funded by Bundesministerium für Bildung und Forschung (01 ISB 04)

While the importance of integrated models and tool processed documents is certainly growing – see model based software development [3] to mention just one example – there is also a fundamental need for the efficient and consistent editing and processing of structured text-based documents. In software engineering, such documents are often demanded by development process models such as the Unified Process [19] or the German V-Modell®[1] [7]. Even agile approaches like eXtreme Programming (XP) require structured text-based documents such as story cards or task cards [1]. As for process engineering, processes can be modeled and documented efficiently in the form of structured text-based documents, because they contain significant amounts of textual data that is generally structured according to a well-defined meta-model. Processes that are documented this way are much easier accessible by process support, enactment or simulation tools, such as XCHIPS [15].

Requirements for editing and processing structured text-based documents in an engineering context are different from those for writing books or articles. Besides commonly accepted requirements such as what-you-see-is-what-you-get-editing (WYSIWYG) and spell-checking, we identified additional requirements that reflect the specific needs of engineers. Our list of requirements was inspired by the lists in [34] and [21]:

- *Support team-based editing (R1)*: In engineering projects, usually groups rather than individuals work on documents. Parallel editing and feedback about the impact of changes are important necessities in a groupware environment [13]. In process engineering, group-editing capabilities are particularly important during initial process documentation and during organization and project specific process adaptations, because the editing involves an abundance of domain, application, process and organization experts all contributing at the same time to the same result.
- *Assure structure and consistency (R2)*: Text-based engineering documents mostly have a rigid and well-defined internal structure. Maintaining the structure and internal consistency is of paramount importance for the quality of the result and for efficient handling. Consistency often exceeds the single document and spans a set of different, interconnected, concurrently edited documents. Fixed structure and consistency are also important for process models that are input for project support and process simulation tools, for instance.
- *Enable post processing (R3)*: In engineering projects, people and tools often need to access and process the information contained in the work documents. This is necessary, for instance, to generate customized representations of a document's contents. Tools must be able to access and query the document contents. Processes and process models, for instance, can be automatically enacted or simulated by tools, provided the process information is present in a post-processable format and structure [15].

In general, tool support for editing and processing *integrated models* and *tool processed documents* has steadily improved over the last years (cf. [12], [16], [17], [33]). Configuration management systems, such as CVS, Subversion, PVCS™, and Microsoft Visual SourceSafe™, have proven their usefulness and efficacy for team-based editing (R1) in many years. They enable distributed, parallel editing of source files with automatic merging of concurrent changes.

Working on *structured, text-based documents* in distributed and parallel work sessions, however, still yields a variety of problems. Some, often only theoretical answers, are provided by the research field "Computer Supported Collaborative Work (CSCW)" [4] which offers a wide variety of different approaches. However, only some have proven in practice.

On the one hand, there are solutions based on the comparison and integration of several Microsoft Office Documents, possibly exchanged via a central server. Because the documents themselves are quite complex, this integration is a rather manual and cumbersome task. Moreover, common experiences with Microsoft Word show vital problems in handling large files (i.e. documents with more than 200 pages). Therefore, this solution is only suitable for smaller projects.

On the other hand, there are proven approaches based on text files and corresponding version management systems. Many very large documents have been successfully written in this way in distributed teams based on a combination of T_EX [20] and the versioning system CVS [10], for instance. Hereby, many conflicts usually arising during the process of writing can be automatically resolved by the logic of the CVS merging algorithm and hence do not require user involvement.

[1] V-Modell is a registered trademark of the Federal Republic of Germany.

This approach appears to be a promising starting point for tools working with *structured, text-based documents*: TEX typesetting files can be edited concurrently by multiple users [28] and merged by configuration management or versioning tools. Although TEX-based solutions hence fulfill the requirement R1 they have some drawbacks such as a lack of WYSIWYG editing (common requirement), assurance of structure and consistency by editing tools (R2), and post processing by other tools than the TEX processor itself (R3).

With the advent of XML [35] as structured format for data exchange, multiple tools have been developed to edit and manage XML files (R2, R3). The XML Spy toolsuite [30] is an example for powerful XML editing and related tasks. Compared to other XML editors, the user is not required to edit the textual representation directly and can alter the content using forms instead. However, these forms need to be defined beforehand, which is a drawback in case of modifications to the XML structure. In this paper we describe 4everedit as an alternative XML editor that provides the additional flexibility needed – in this case the generic form generation mechanism.

The exchange and consistent update of XML files (R1, R2) is sometimes handled using XML databases (cf. [32]). This appears to be a rather sophisticated and error-prone solution; furthermore the technique itself has not yet matured. The combination of proven version management tools, such as CVS [10], and XML editing failed – at least at the time of evaluation – due to the inability of CVS (and even its inofficial successor subversion) to include effective comparison and merging tools for this format. The approach in [11], for instance, combines XML and CVS but requires a non-standard file comparison solution. Therefore, the evaluated approaches lacked flexibility for at least one of the requirements integration with standard CVS (R1), consistency checking (R2) and post-processing (R3). The same arguments also hold true for professional process modeling tools and powerful graphical and/or web-based tools such as SPEARMINT [15] and the Rational Process Workbench [30].

Summing up, editing and processing structured text-based documents is important. Although viable solutions for the engineering specific requirements R1 to R3 exist for *integrated models* and *tool processed documents*, they have not been successfully and systematically transferred to editing and processing *structured text-based documents*. Such documents are currently often edited with standard text processing tools such as Microsoft Word™ or Open Office, which support efficient single user text editing and formatting but lack the ability to fulfill the requirements R1 to R3 [21].

Because of these reasons, we developed new concepts to support team-based editing of *structured text-based documents* with respect to the specific needs of an engineering environment (R1 to R3). We realized these concepts in the form of the tool 4everedit [24] after evaluating a variety of alternatives. The 4everedit editor has been successfully applied within a large process modeling project - a case study that is discussed in the next section.

The subsequent content of this paper is organized as follows. The next section describes settings of the project as well as the case study. Section 3 introduces the concepts and architecture of 4everedit. Hereby we focus on the technical challenges introduced by the desired editing and processing concept. Section 4 presents the results of applying 4everedit in a context of creating and editing a software development process model. We show the advantages of our approach by giving expressive statistics and further analysis. We particularly show where the proposed approach gives a benefit compared to traditional ways of structured text-based document editing and share our lessons learned. Finally, a short conclusion and outlook in Section 5 completes the paper.

The main contribution of this paper is to show a successful real-world application of a tool for team-based editing of structured, text-based documents and in particular for process documentation management. We present our approach based on an analysis of the requirements and report on experience of applying the tool in a project of significant size in an industrial context. The project is representative for many similar challenges of team-based document editing. The reported experiences indicate successful application and might inspire and encourage similar solutions.

2. Project Case Study

The V-Modell® is the German standard model for system development and lifecycle processes conducted within engineering projects of federal administration and defense. It is publicly available and many companies have successfully adopted it to their needs. The V-Modell regulates the system

development and maintenance process; it defines binding sets of activities and artifacts, and accompanying processes such as quality assurance, configuration management, and technical project management [7].

In 2002, the most recent update of this standard dated back to 1997. Many innovations and best-practices of software engineering were lacking and the application, adaptation and extension of the V-Modell were often reported to be cumbersome.

Consequently, a project was initiated [29] to analyze the drawbacks of the V-Modell, to identify potential for improvement, and finally to develop a new redesigned version called V-Modell® XT – to be established within the federal administration, military and cooperating companies [8]. Key properties of the project – our case study – were [29]:

- Project duration: October 2002 to August 2004
- Team members: more than 35 individuals
- Team locations: 8 organizations
- QA: over 15 external reviewers
- Effort: over 30 person years

As the numbers illustrate, the project is considered a large process engineering project. Editing and processing structured, text-based documents was an important challenge during the course of the project. Various structured text-based documents had to be created, edited, post processed, and maintained. Naturally, the most important was the main project result: the V-Modell XT process documentation itself.

The initially estimated size of the printed version of the V-Modell XT process documentation was 400 to 800 pages, containing strongly structured and cross-referenced text, enriched by figures and tables. Required deliverables included a printable format (PDF) and versions in HTML and Microsoft Word™. The process description was demanded to support extraction and modification by tools for on-site process tailoring and project support.

One of the most vital design decisions stems from the initial analysis of the V-Modell's drawbacks and improvement potentials: a meta-model was specified and documented to precisely define and relate the fundamental concepts of the new standard [5]. This is used as a process definition language, such as the SPEM [27] or to the Product- and Process-Language described in [14]. The meta-model served both for the communication of the various concepts and as a guideline for the process design phase:

- Duration: July 2003 to August 2004
- Editors: 26 individuals
- Editors locations: 5 organizations
- QA: over 15 external reviewers
- Effort: over 20 person years

Because the process language was developed from scratch and first of all used for the V-Modell, we expected the language itself to improve over time. Hence, besides the general requirements for editing and processing structured text-based documents in an engineering environment (cf. requirement R1 to R3) we identified an additional, project specific requirement:

- *Modifiable document structure (R4)*: the anticipation of changes to the process description language (meta-model) requires that the structure of the process documentation needs to be flexible and modifiable throughout the project. Changes must be feasible in a consistent and effective way. Tool support must reflect this possibility of change.

The challenging project goals as well as the size and distribution of the project team led to high level project risks. One particular risk was that the editing team would not be able to produce a consistent process document within the given time-frame, because the team was large, inhomogeneous, spread out over multiple locations and the work result was expected to be of high complexity with sophisticated internal cross-references and consistency requirements.

To mitigate especially this high risk, we provided a reliable editing and processing tool for structured text-based documents that establishes an appropriate and powerful solution for the proposed requirements R1 to R4. Furthermore, we defined an operating procedure (a process) for concurrent editing and integration that was applied by the editing team.

The desired tool support strategy and the editing process were similar to "continuous integration" as proposed in agile software development processes [1]. From the beginning of the project, all editors were required to work with a central server-based file repository and to integrate their local changes at least once a day. We made sure that every user was able to generate a printable version of the process documentation using the same formatting to verify any changes locally and instantaneously. In regular intervals, a full export consisting of the process model in all different output formats was created automatically on the server and published for inter-

nal team review. Every eight weeks, the most recent build in production quality and layout was submitted to external reviewers as a release for quality assurance. In this aspect, the development process was quite similar to that of software: nightly build and nightly tests correspond to checking the various consistency constraints and creating a human readable output for peer review.

Project requirements and identified risks required an ambitious and innovative approach to deliver a result in the given time-frame with the expected quality; at the same time the project risks had to be kept at an acceptable, controllable level. Our solution after a thorough analysis of options (for a selection, see Section 1) was the 4everedit editor and platform that we developed and customized with respect to the specific needs of our project. We developed 4everedit based on an existing open source editor and XML framework that we developed in a previous research project [25].

3. 4everedit architecture

The identified requirements (R1 to R4) demand fully team-based editing of large, structured text-based documents with intricate internal references and dependencies. The tool *4everedit* is a generic XML based editor with a dynamic user interface, team-support through integration with the versioning system CVS [10] and a mechanism for post-processing the edited documents.

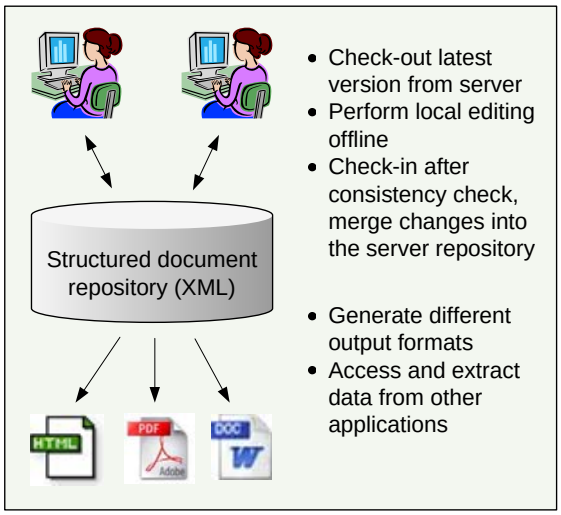

Figure 1: 4everedit usage scenario

Figure 1 shows the usage scenario of 4everedit. Multiple users access a central structured data repository to check out consistent versions of the document of interest and to edit them independently and locally. Users commit changes back to the repository. Provided a set of consistency checks passes, the changes are merged into the repository. This mechanism enables users to concurrently edit documents without any particular locking mechanism.

In case of consistency violations or merge problems, the server responds with a descriptive error message and requires the user to solve any problems first before continuing. Such errors neither harm the consistency of the server based document repository, nor block other users from editing their versions of the document and checking them in whenever they choose. The contents in the document repository are thus always kept consistent and up-to-date. Because the data within the repository is structured and consistent, other applications can query and extract it for visualization or further processing.

The following sections explain in detail in which way the four main requirements are met by our solution and why our tool is well suited for process modeling and subsequent process support and simulation. We start with the project specific requirement R4, because it allows us to introduce the basic tool properties first.

3.1. Modifiable document structure (R4)

A very important requirement in our project setting was the ability to modify the internal document structure during the runtime of the project in order to adjust it to a possibly changed process metamodel. 4everedit is a customized generic XML editor similar to the popular XML Spy [37], for instance. For any edited file, it requires a document structure definition in form of a XML schema. The schema defines the structure of the work document, all of its elements as well as their arrangement within the file, and finally constraints with respect to internal cross references between document elements. Thus the schema determines the operations which are permitted on the document (e.g. add, delete, link element etc).

The user interface is similarly flexible to support a generic adaptability to different document structures. Figure 2 shows a screenshot of the tool's generic user interface. The left side depicts the XML structure in form of a tree and lets the user modify the structure by adding, moving or removing elements. The right side shows the detail view

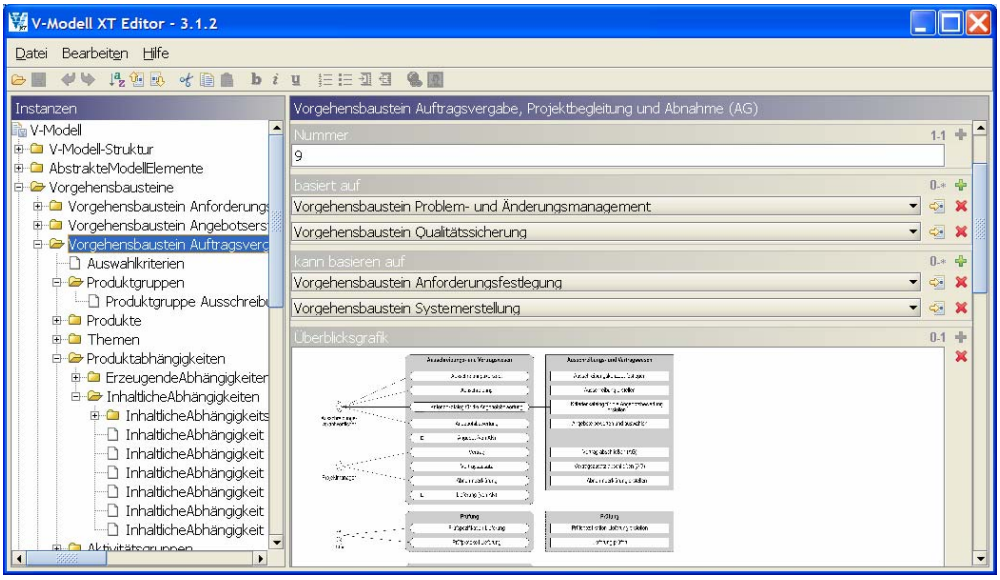

Figure 2: 4everedit generic user interface

of a particular element selected in the tree with generic input fields of different types, such as plain text, rich text (including images), references, choice values and so on. 4everedit displays the document structure and details and only permits valid changes as defined by the schema. For instance, it only offers to add or delete elements if the document schema provides for it. The dynamic user interface supports cross-references within the document. They are declared in the schema and presented as drop-down-boxes containing an on-the-spot composed list of possible reference targets.

4everedit is also very robust in reading the XML document. Any XML file that conforms to a schema will be parsed, no matter what the current formatting is or which comments are present (cf. Section 3.2 for the XML file format). Additionally, even missing and thus schema-violating XML elements and attributes are tolerated and communicated to the user during the succeeding run of the various consistency checks. This makes it very easy and efficient to perform manual file edits, which are sometimes necessary after a CVS merge conflict or after an XML schema change.

3.2. Support team-based editing (R1)

Editing files and documents concurrently in a team requires special workflows, data formats, and tool functionality. We use XML [35] as persistence format for our documents and benefit from its inherent structure and text based file format.

Intuitively, requirement R1 is fulfilled by using a configuration management tool such as CVS for the central repository. CVS is able to merge different versions of the same text file to a new version if there are no mutual changes of the same parts of the file. CVS relies on the UNIX tool diff [18] to identify changes and merge documents.

However, there are several issues and problems related to this solution; in the following we describe the problems with comparing and merging XML files and show our solution. A common difficulty when processing XML documents as text files is that one XML document can have different file representations that are equal in meaning. Therefore, 4everedit saves XML documents in a normalized, unambiguous way. The output is fitted particularly to line based file comparison algorithms like that of CVS. Content data format, element attribute ordering and whitespace placement follows clear rules. The chosen format is also very human legible to make testing and manual XML manipulation easy.

A challenging problem when comparing XML files line by line is the high ambiguity due to their very regular build-up with a high number of often consecutive repeating lines or elements; this is a difference compared to source code files. It prevents the standard file comparison algorithms as used by CVS from working effectively. There exist effective solutions for comparing two XML files for differences (see e.g. [9], [11]). We cannot use those, however, because we chose to use a standard

CVS server as found and centrally operated in numerous organizations; therefore we had to rely on its built-in line based file comparison algorithm. We illustrate the problem and our solution with the following example:

```
<Process>
 <Products>
  <Product id="prod1">
   <Name>Project Handbook</Name>
   <Desc>Project goals</Desc>
  </Product>
 </Products>
</Process>
```

The XML file above represents a process instance with a set of work products as defined by the `Product` element. Each work product has a name, an optional link to an activity that creates it (not shown in the file above) and a description. A process should be documented in a team in the form of the above XML. Team member Alice is responsible for a product called "Project Handbook" and Bob is responsible for managing the associations between products and activities. Both work in parallel and independent of each other on the process model. Alice changes the name of the work product resulting in the following locally changed XML fragment:

```
  <Product id="prod1">
   <Name>Project Manual</Name>
   <Desc>Project goals</Desc>
  </Product>
```

Bob concurrently introduces a reference (or link) from the product "Project Handbook" to an activity that creates the product. He does so by adding a 4everedit reference element `ActivityRef` that results in the following XML fragment:

```
  <Product id="prod1">
   <Name>Project Handbook</Name>
   <ActivityRef link="act1"/>
   <Desc>Project goals</Desc>
  </Product>
```

Alice successfully commits her work to the CVS server first. Meanwhile Bob has finished his work and tries to commit his local changes. Because he is second, he gets a message from the CVS server that he has to merge the current version on the server into his local version. The CVS client tries to perform the merge automatically. However, Bob gets the following merge conflict:

```
  <Product id="prod1">
<<<<<<< process-model.xml
   <Name>Project Handbook</Name>
   <ActivityRef link="act1"/>
=======
   <Name>Project Manual</Name>
>>>>>>> 1.65
   <Desc>Project goals</Desc>
  </Product>
```

The line based comparison algorithm used by CVS cannot separate the two changes. It treats file differences as sequences of line deletes and inserts [18]. To avoid ambiguities, 4everedit assists CVS by annotating the XML structure with specific comments containing element ids to make all lines unique. This and similar mechanisms allow CVS to isolate modifications from different sources and track them down to specific elements, respectively lines, in the XML. The following XML fragment shows the annotated XML of the original document, as saved by 4everedit:

```
 <!-- Products Start -->
 <Products>
  <!-- Product Start prod1 -->
  <Product id="prod1">
   <!-- Name Start prod1 -->
   <Name>Project Handbook</Name><!-- prod1
-->
   <!-- Name End prod1 -->
   <!-- ActivityRef Start prod1 -->
   <!-- ActivityRef End prod1 -->
   <!-- Desc Start prod1 -->
   <Desc>Project goals</Desc><!-- prod1 -->
   <!-- Desc End prod1 -->
  </Product><!-- prod1 -->
  <!-- Product End prod1 -->
 </Products>
 <!-- Products End -->
```

With the XML annotations, our example scenario leads to a correct automatic merge for Bob that he can now commit without problems and without ever seeing XML markup. CVS keeps both changes separate and automatically merges them correctly, without a conflict. The resulting XML fragment is shown below:

```
  <Product id="prod1">
   <!-- Name Start prod1 -->
   <Name>Project Manual</Name><!-- prod1-->
   <!-- Name End prod1 -->
   <!-- ActivityRef Start prod1 -->
   <ActivityRef link="act1"/><!-- prod1 -->
   <!-- ActivityRef End prod1 -->
   <!-- Desc Start prod1 -->
   <Desc>Project goals</Desc><!-- prod1 -->
   <!-- Desc End prod1 -->
  </Product><!-- prod1 -->
```

Without annotating the XML, similar problems occur for many other team-based usage scenarios, such as concurrently changing, deleting and moving elements. By extending the XML file with the described annotations, CVS can automatically merge most of these concurrent changes correctly. Concurrent changes in the exact same location of an XML file, however, always result in an inevitable merge conflict, which must be corrected manually. This can be kept to a minimum though, if users are assigned (different) elements that they are allowed to edit.

3.3. Assure structure and consistency (R2)

An integral part of working in a team on a central data repository is to ensure the repository's internal structure and referential integrity. 4everedit addresses both issues as follows. It guarantees the internal document structure using XML schema validation, which is a standard functionality of any XML editor. 4everedit allows for structure alterations only in accordance with the associated XML schema. Additionally, 4everedit detects any schema violations introduced by editing the document outside of the editor when the document is loaded. In this case, the responsible user must correct all errors before editing can continue.

Ensuring the document's internal referential integrity requires mechanisms on a semantic level that take the document's contents and application domain into account. A domain expert must provide application-specific meanings to the document's contents. 4everedit supports the mechanism of pluggable consistency checks for checking and guaranteeing a document's referential integrity and semantic consistency. Currently, consistency checks are plug-in modules implementing consistency rules. They can access the document's complete data model using 4everedit's standard data access API to detect any violations of consistency rules. This implementation was driven by simplicity. For a discussion about shortcomings and alternatives, see Section 4.3.

Consistency checks, for instance, can detect whether all document elements are referenced at least once, or that a graph of dependent elements, e.g. activities, contains no cycles and is fully connected and reachable from a root element. In process engineering, it is very important to keep the process documentation in a consistent state and to detect any violations immediately, so that dependent tools, e.g. for process simulation, can operate on a consistent model. The mere size of many process documentations makes an effective automated consistency assurance mechanism mandatory.

4everedit executes consistency checks on the client and on the server. On the client, consistency violations are checked for every time the document is saved. They can be ignored by the user as long as the XML document remains well-formed and all internal references have valid targets. This enables a much smoother editing process with intermediate inconsistent states. On the server, however, consistency is always enforced. 4everedit does not allow inconsistent documents to be committed to the server's repository.

Besides consistency checks, 4everedit provides a second important structure and consistency enforcement mechanism that is located on the repository server. Chosen parts of the team-edited document can be locked ("frozen") and prevented from modifications. Figure 3 depicts the entire document checking workflow on the server, all triggered by a CVS commitinfo script [10]. The mechanism is as simple as powerful:

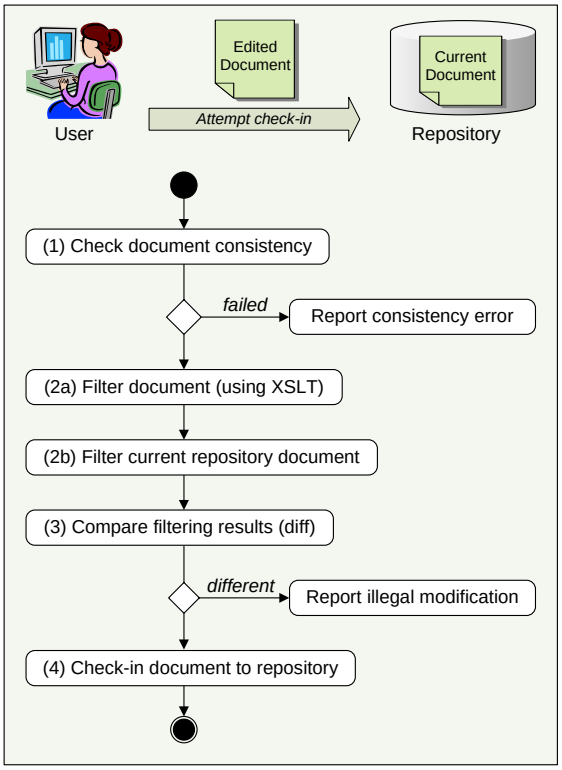

Figure 3: Document checking workflow on the repository server

After checking the document's consistency (1), the server performs a document modification check. An XSL transformation filters out all non-locked parts of the XML document. Thus, the transformation result contains only the locked parts, those that shall remain unmodified.

This filtering step is applied to both the document that is subject to check-in (2a) and the most recent document in the CVS repository (2b). If both transformation results are equal (3) – as compared by the standard UNIX diff tool – no modifications to any locked element did happen and the check-in can proceed (4). Otherwise, the user will get an error message pointing to the XML elements that caused the conflict.

An authorized user (e.g. the document manager) controls the locking ("freezing") scheme by manipulating the XSLT script that performs the filtering. Any client with access to the CVS and sufficient authorization can change the script, because it is also located in the CVS repository. The document modification check can make use of all the power and flexibility of XSL transformations. This ranges from "freezing" individual elements (e.g. a chapter named "Introduction"), all elements of a certain type (e.g. all top level chapters of a document), certain attributes of all elements (e.g. all element names), relations to other elements and any combination thereof to freezing the entire document but for certain exceptions. Freezing can apply to textual contents as well as to the XML structure only; the latter is very helpful to keep a basic document build-up intact and still permit modification of the textual parts. Freezing could also distinguish different users or editing roles for a fine-grained modification authorization scheme – however, we never implemented that. It is easy to create more complex checks, such as freezing certain elements but leaving the order in which they occur open, fixing the number of elements of a certain type etc. The XSLT language [36] with its Xpath expressions provides very powerful means to quickly implement such checks and transformations in scripts of minimal length.

Thus, a document manager can gradually lock the document or parts thereof and restrict the collaborative editing to parts that have not yet been finished and reviewed.

3.4. Enable post processing (R3)

An advantage of working with rigidly structured documents is the ability to further process them with different tools for extracting and querying data, and for rearranging and visualizing the contents. 4everedit can post-process the work document by integrating XSLT processors and document creation tools. A three level post-processing pipeline following a simple pipes-and-filters architecture [6] is controlled from a configuration file, as depicted in Figure 4.

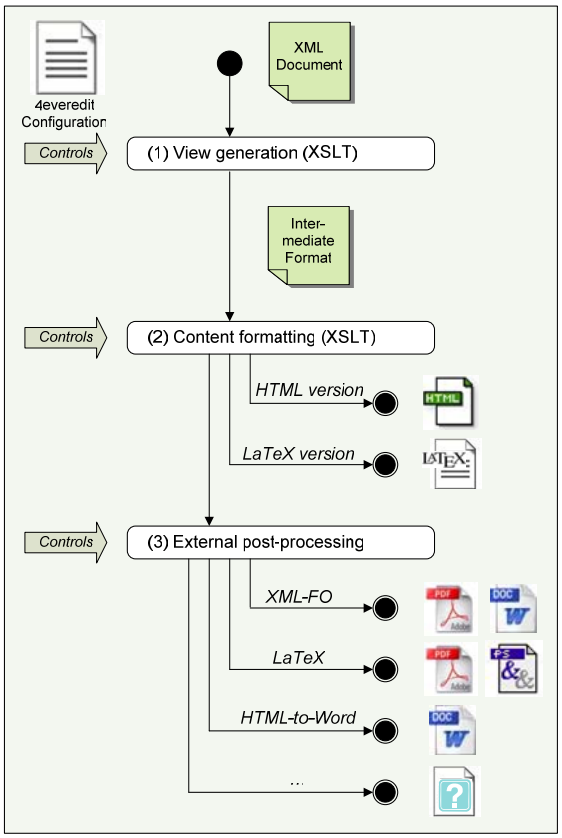

Figure 4: Post-processing pipeline

First, in the view generation phase, 4everedit transforms the document into a presentation and content independent, intermediate format. Second, in the formatting phase, 4everedit transforms the result of step one further into an output format, such as HTML, XML-FO, LaTeX, docbook etc. Finally, if needed, 4everedit invokes external post-processors that transform the output of the second step into the final result. Post-processors include LaTeX compilers and XML-FO processors, such as Apache FOP, XEP and JFOR, to generate PDF, PostScript, RTF and many other output formats.

4everedit can use the post-processing pipeline, for instance, to generate Electronic Process Guides (EPGs, cf. [2]) in hypertext format to provide proc-

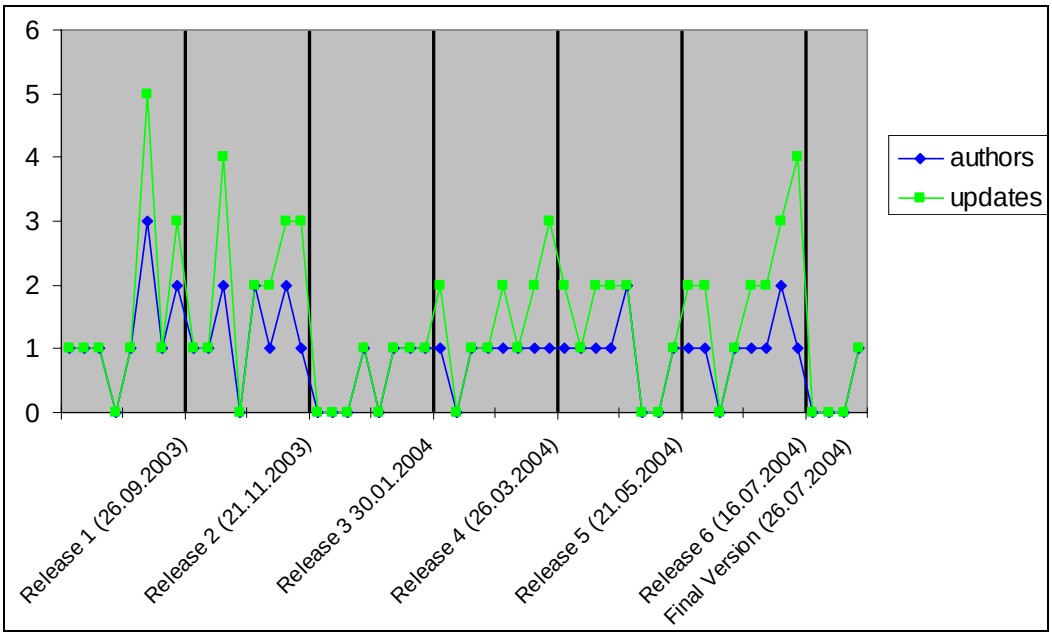

Figure 5: Document structure modifications

ess guidance for developers. For project management, 4everedit can extract structural information from the working document that shows the percentage of completion of the work in progress.
By using standardized intermediate formats in the chain of post-processing, it is very easy to connect external tools to generate all kinds of desired output formats. It furthermore decouples the separate steps in the chain and keeps them easy to develop, modify and test.

4. Experiences with applying 4everedit to process documentation management

To edit, maintain and process the V-Modell XT in the described project case, we implemented and used 4everedit with respect to the engineering specific requirements R1 to R4. In this section we will report our experiences with applying 4everedit. For each discussed requirement and corresponding solution, we present experiences and findings. The following sections demonstrate how the four main requirements are fulfilled by 4everedit. We cover the requirements in the same order as in Section 3.

4.1. Modifiable document structure (R4)

When the team started editing, we expected that the structure of the edited process documentation would change over time. Figure 5 shows that structural changes indeed occurred during the entire project, 72 times in total. The diagram shows the changes (XML schema modifications) per week and the number of different authors performing the changes. Compared to the number of overall modifications of the XML document (2,337), the number of structural changes (72) could be kept rather low and could be performed by a small group of experienced engineers. This was the foundation for document handling with both a modifiable structure and short turn-around times. Because 4everedit is generic in its user interface and XML schema, no code changes were required when we changed the document structure.

In those cases where we added details to the document structure and thus simply refined it, we could leave the document unchanged. In about a third of all changes, we had to migrate the document's content according to the modified schema. 4everedit does not provide any built-in support for automatic content migration; we had to migrate the document manually. This was inevitable and required an exclusive lock of the document. Thus, we tried to schedule it for weekends or nights. Mostly the migration was simple and thus was carried out quickly by hand. About ten times it was necessary to write a simple XSL transformation script. The largest of these scripts was about 110 lines of code, including comments and empty lines. We consider

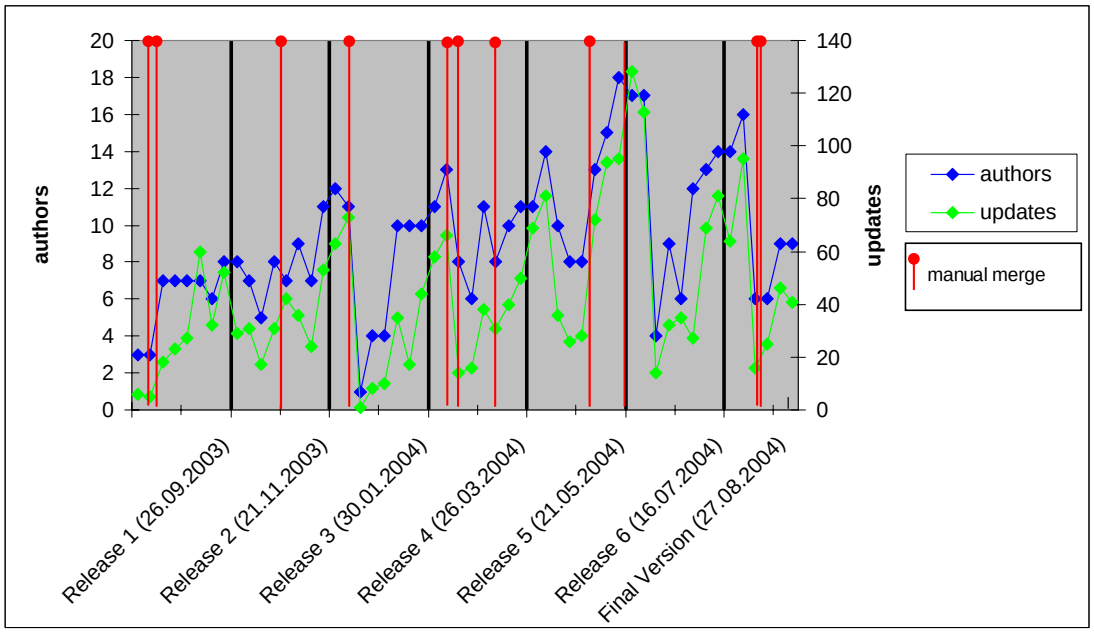

Figure 6: Document versions and authors

the effort spent to implement and test these scripts very acceptable; none required more than two hours of work and most often, parts of existing scripts could be reused. The aforementioned robustness of 4everedit in parsing XML files combined with an XML validator and consistency checker that provided us with telling error messages complemented the ease of use in the content migration scenario.

To sum up, 4everedit provided sufficient and very valuable support and flexibility for document structure modifications – even for a project of considerable size and with inevitable manual interactions. This support eliminated a previously existing concern in the team that this approach of robust XML handling with sporadic manual interventions but without defined automatic migration procedures would not scale to our project size. Nevertheless, we are interested in further research about automating migration steps in this scenario.

4.2. Support team-based editing (R1)

We had identified team-support as a crucial success factor for the project. To support team-based editing, we designed 4everedit to use a commented XML file that is stored in a central CVS repository. Because we could not find any experiences or representative transferable test results, this seemed to us to be the most critical part during the application of 4everedit in the project.

However, the final evaluation results are very telling, as shown in Figure 6: during the project, the authors created 2,337 versions of the V-Modell XT XML file. A number of 26 different authors concurrently edited this XML file. The maximum number of versions that were created in a day is 36 and the maximum number of different authors that modified the document on a single day is 13. The diagram shows the number of new versions each week and the number of different authors creating them.

These numbers show that the evaluated project case serves as a load test environment for the proposed solution for team-based editing. Nevertheless, 12 times a manual integration of the XML file were required because of CVS conflicts. With a percentage of 99.5, the update and merge mechanisms of CVS were sufficient enough to integrate the concurrent editors' work. Hence, the presented technique – a commented XML file – worked very well to enable CVS to handle concurrent editing, and even exceeded our initial expectations.

4.3. Assure structure and consistency (R2)

The integrity of structure and consistency was never jeopardized during the project – mostly ensured by over 70 consistency checks that we had implemented at the end of the project. We could keep the V-Modell XML file consistent throughout the entire project and thus were always able to edit and process the document. We always had confidence in its inner structure, even after major rearrangements of the file's build-up.

However, maintaining the implementation of the consistency checks was a difficult task. Whenever the structure of the document changed, the affected consistency checks needed a rework. The reason was mainly 4everedit's interface for the plug-ins. The interface to access the document's contents from the consistency checks was too close to the actual XML structure. Moreover, the consistency checks were imperatively implemented in Java instead of, for instance, formulating them declaratively as consistency rules. In the next versions of the editor, the consistency check plug-in mechanism will be replaced by a rule engine, such as JRule or xlinkit [26]. Combining this with a use of XML metadata and the Resource Description Framework (RDF) [31] that emerged in the context of the Semantic Web, should give us enough flexibility and a higher degree of decoupling of data, schema and consistency rules.

We implemented the document "freezing" mechanism in an effort to stabilize the contents of a document of growing size and sophisticated structure. Even with an editing operating procedure in place and a change control board restricting the permitted edits to certain areas in the document, it was not possible to control and trace all modifications – some just happened accidentally. The freezing mechanism solved that problem. Due to its technical simplicity and the use of standard tools, we could implement it in less than a day. Modifications, extensions and temporary disable operations of the XSLT "freezing" script were a matter of minutes.

In retrospective, the mechanism proved invaluable in gradually stabilizing the document and its key elements early on in the process till the end. First, the team decided about the key elements and their names (the architecture of the process documentation), so we could freeze this information. Second, the responsible users could define the basic attributes of these elements and their relations to or dependencies on other elements. Consequently, we fixed this information next. And third, the editing users could finish the textual and graphical parts successively, let them freeze and give them into review, as planned in the release plan. These steps happened at different times for elements of different levels of detail and priority. Of course, several times we also had to unfreeze certain elements because a rework was requested by the reviewers or decided by the change control board.

Thus, the freezing mechanism provided an effective means for our quality assurance process: we could give frozen parts into review and be sure about their status and quality from then on. The percentage of frozen versus modifiable elements in the document provided a telling metrics for project management and indicated the degree of completion of the entire document.

Figure 7 summarizes our document editing and quality assurance process. A document management group planned in accordance with the editing team the document's architecture as well as its release plan and decided on any structural changes. A technical team performed the structural changes consistently when requested and resolved the few occurring CVS integration conflicts. The numerous editing users could work collaboratively and independently on their local workstations on an always consistent central model.

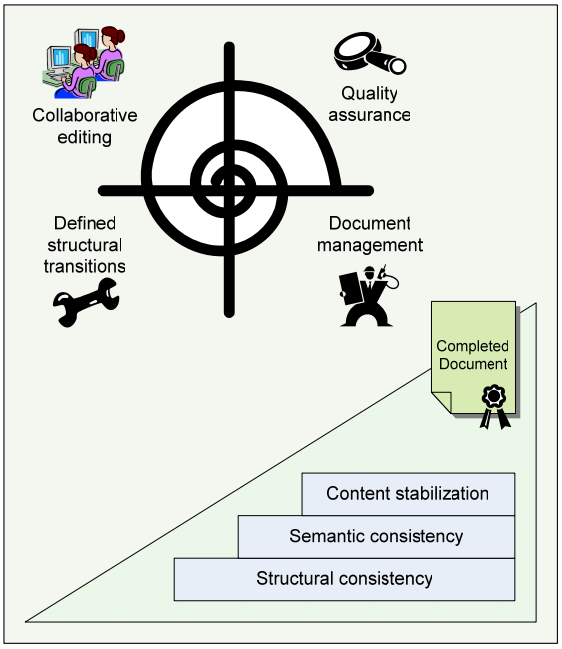

Figure 7: Collaborative document editing using 4everedit

A multi-tier review team performed reviews of parts of the document and of the entire document in fixed intervals. All this happened incrementally and in a very agile way based on the strong support that 4everedit offered: assuring the document's structural and referential consistency, enforcing the semantic consistency, and stabilizing the work results by freezing the document's contents. The described combination of strategies, procedures and technologies made the timely delivery of the V-Modell XT possible.

4.4. Enable post processing (R3)

4everedit provides a very flexible post processing framework. Early in the project we generated a 146 page PDF file out of an 881 KB XML file. At the end of the project we were able to generate the following documents out of a four megabytes XML file and more than 130 external GIF images:

- a fully cross-referenced, indexed and bookmarked PDF file with 635 pages and a size of 34 MB
- the similar content split into nine separate PDF files
- complete and separate Microsoft Word files
- an HTML-based version with over 1,250 files
- Word templates for 76 defined products

The look and feel of the output changed significantly during the project. Starting out with a table- and icon-based design, we finally delivered a highly structured text-based document with a sophisticated and fine tuned design and a high number of internal hyperlinks and generated indices.

Generating the PDF results led us astray for a while. A long time, we generated XML-FO and post-processed this into PDF, using Apache FOP. This produced fairly good results but not the envisioned deliverable quality, with a powerful paragraph control, for instance. Also, the generation process with FOP used more than 5 GB of RAM memory on a UNIX server for the full export. Thus, we switched to the generation of LaTeX source files and their compilation to PDF; we instantly gained professional book-quality layout and almost unlimited flexibility and scalability. This switch required the development of a similar XSLT script for LaTeX. A student implemented this within a few weeks – fortunately he could make good use of the extensive experiences manifested in the XML-FO generation script. LaTeX also significantly simplified the script because we got all heading numbering and table-of-content handling for free.

At the end of the project we had created more than 70 versions of the PDF generation script alone. Furthermore, the maintenance of the intermediate format generation scripts (step 1 in Figure 4) was very complex and error prone. Whenever the structure of the document changed, these scripts had to be checked and often adapted or reworked. The first versions of the generation scripts lacked a flexible architecture to support structural changes. This improved significantly during the course of the project; we used modularization and abstraction whenever possible. In the most recent versions, we even provided the flexibility for the process editors to define the outline of the generated results in the editor itself – and even supported different variants (views) referencing the same source data. 4everedit would pass user selected, configurable options to the post-processing scripts that were flexible enough to dynamically compose the data on the fly as requested.

Currently, we are reworking the generation scripts towards a more generic configurable pipes-and-filters architecture. This will make maintenance and improvement of the generation scripts even easier in future.

5. Conclusion and Future work

Team-based editing of structured text-based documents is an integral task in engineering projects. In particular, it is an essential task for managing process documentations. Existing tools, however, do not support the specific requirements for team-based editing of many engineering documents. All evaluated tools failed to fulfill all our requirements; mostly to support the requirement to be stable against changes of the underlying structure (R4).

For that reason, we have developed and implemented 4everedit and successfully applied it in a large process engineering project. Over a year, 26 editors, from more than five companies, have concurrently elaborated process documentation with 635 pages and 34 MB in size using 4everedit. This impressively proves the applicability of 4everedit for process modeling and process documentation management and the fulfillment of the requirements listed in Section 1. 4everedit has been published as an open source project under the CPL license [24].

A particular elegance of the 4everedit approach lies within its simplicity. Based upon proven solutions such as CVS with its line-based diff and merge utilities, a rather small modification in the XML saving process led to a powerful solution, which might serve as a viable basis for the application of existing process support, enactment and simulation tools, such as XCHIPS [15] or inStep [23].

In the meantime, 4everedit has been successfully applied within other industrial projects. It has been used to edit and process requirements documents, project documentations, software architecture documents, tool configurations and business process definitions. This shows that the presented results from our process engineering project can be transferred to other engineering disciplines.

Currently, we are integrating a rule engine to ease creation and maintenance of the consistency checks. In the future, the post processing framework will be redesigned to provide a more flexible and adaptable architecture. In the long term, more research needs to be done in the area of model-based editing of structured text-based documents and in particular process documentations. Our project shows that the size of information we have produced and processed was approaching the manageable maximum. For larger documents, more sophisticated modeling techniques and methods must be developed and implemented.

Acknowledgments

We are grateful to Christian Bartelt, Dirk Niebuhr, and Tim Schumann for comments on earlier versions of this paper. We thank the anonymous reviewers for helpful comments to improve this paper. Moreover we thank the whole WEIT team for the great and fun experience of working together.

References

[1] K. Beck, *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 1999.

[2] U. Becker-Kornstaedt, "*Der V-Modell Guide: Webbasierte Unterstützung eines Prozeß-Standards*", Workshop of the GI-Fachgruppe 5.1.1 Vorgehensmodelle – Prozessverbesserung und Qualitätsmanagement, 1999.

[3] S. Beydeda, M. Book, and V. Gruhn (eds.), *Model-Driven Software Development*, Springer, 2005.

[4] U. Borghoff, and J. Schlichter, *Computer-Supported Cooperative Work - Introduction to Distributed Applications*, Springer 2000.

[5] M. Broy, M. Deubler, M. Gnatz, H. Hummel, W. Kranz, M. Meisinger, D. Rauh, A. Rausch, and W. Russwurm, *Abschlussbericht WEIT Phase I*, http://www.v-modell-xt.de/ergebnisse1.html, 2003.

[6] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *A System of Patterns. Pattern-Oriented Software Architecture*. Wiley, 1996.

[7] BWB-IT I-5, AU 250, Entwicklungsstandard für IT-Systeme des Bundes, Vorgehensmodell, 1997.

[8] BWB-IT I-5, *Technische Aufgabenbeschreibung zur Weiterentwicklung des Entwicklungsstandards für IT-Systeme des Bundes auf Basis des V Modell 97*, Sept. 2002.

[9] G. Cobena, S. Abiteboul, and A. Marian, „Detecting Changes in XML Documents", In *Proceedings of the 18th International Conference on Data Engineering*, 2002.

[10] Concurrent Versions System (CVS), website, http://www.cvshome.org, 2004.

[11] B. Delinchant, L. Gerbaud, F. Wurtz, and E. Atienza, "Concurrent design versioning system, based on XML file", In *28th Annual Conference of the Industrial Electronics Society (IECON)*, IEEE, 2002.

[12] P. Dewan, and J. Riedl, "Towards computer-supported concurrent software engineering", *IEEE Computer*, 26(1):17-27, January 1993.

[13] C. Ellis, S. Gibbs, and G. Rein, "Groupware: Some issues and experiences", *Communications of the ACM*, 34(1):39-58, 1991.

[14] M. Gnatz, M. Deubler, M. Meisinger, and A. Rausch, "Towards an Integration of Process Modeling and Project Planning", *Proceedings of the 5th International Workshop on Software Process Simulation and Modeling (ProSim 2004)* of ICSE 2004, 2004.

[15] I. Grützner, J. Münch, A. Fernandez, and B. Garzaldeen, "Guided Support for Collaborative Modeling, Enactment and Simulation of Software Development Processes", *Proc. of the 4th Int. Workshop on Software Process Simulation and Modeling (ProSim)* of ICSE 2003, 2003.

[16] J.C. Grundy, and J.G. Hosking, "Software Tools", *Wiley Encyclopedia of Software Engineering*, 2nd Edition, Wiley Interscience, December 2001.

[17] J.C. Grundy, J.G. Hosking, and W.B. Mugridge, "Coordinating distributed software development projects with integrated process modelling and enactment environments", *Proc. of the 7th IEEE Workshop on Enabling Technologies*, 1998.

[18] J.D. Hunt, and M.D. McIlroy, "An Algorithm for Differential File Comparison", *Computer Science Technical Report #41*, Bell Telephone Laboratories, 1976.

[19] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software Development Process*, Addison-Wesley, 1999.

[20] D.E. Knuth, *The TEXbook*, Addison-Wesley, 1984.

[21] P. B. Lowry, "Design requirements for collaborative writing tools for distributed work over the Internet", In *8th Annual Americas Conference on Information Systems (AMCIS)*, 2002.

[22] M. Meisinger, A. Rausch, M. Deubler, M. Gnatz, U. Hammerschall, I. Küffer, and S. Vogel, "Das V-Modell XT - ein modulares Vorgehensmodell", *11. Workshop der Fachgruppe WI-VM der Gesellschaft für Informatik e.V. (GI)*, Shaker Verlag, 2004.

[23] microTool: in-Step - The Workflow Management System for IT projects. Available at http://www.microtool.de/instep/en/

[24] 4everedit project homepage on sourceforge, http://sourceforge.net/projects/fourever/, 2005.

[25] NOW project homepage, http://now.c-lab.de/

[26] B. Nuseibeh, J. Kramer, A. Finkelstein, "ViewPoints: meaningful relationships are difficult!", In *Proc. of the 25th International Conference on Software Engineering*, 2003.

[27] Object Management Group (OMG), *Software Process Engineering Metamodel (SPEM)*, http://www.omg.org/technology/documents/formal/spem.htm, 2002.

[28] N. Preguiça, J.L. Martins, H. Domingos, J.F. Simão, "System Support for Large Scale Collaborative Applications", *Technical report TR-DI-UNL-01-98*, Dep. de Informática, FCT-UNL, Quinta da Torre, 1998.

[29] Projekt WEIT - Weiterentwicklung des Entwicklungs-standards für IT-Systeme des Bundes auf Basis des V-Modell-97, http://www.v-modell-xt.de, 2003.

[30] Rational Process Workbench, IBM website, http://www.ibm.com/software/awdtools/rup/workbench/

[31] Resource Description Framework (RDF) specification, website, http://www.w3.org/RDF/

[32] A. Salminen, and F.W. Tompa, "Requirements for XML document database systems", In *Proc. of the ACM Symposium on Document Engineering (DocEng '01)*, ACM Press, 2001.

[33] F. Titze, „Improvement of a configuration management system", In *Proceedings of the 22nd International Conference on Software Engineering*, pp. 618-625, 2000.

[34] M. Verlage, B. Dellen, F. Maurer, J. Münch, "A Synthesis of Two Process Support Approaches", In *Proc. of the 8th International Conference on Software Engineering and Knowledge Engineering (SEKE'96)*, 1996.

[35] W3C consortium, Extensible Markup Language (XML) 1.0 specification (third edition), http://www.w3c.org/TR/ 2004/REC-xml-20040204, 2004.

[36] W3C consortium, XSL Transformations (XSLT) 1.0 specification, http://www.w3c.org/TR/xslt, 1999.

[37] XML Spy, Altova, website, Product information available at http://www.altova.com/products_ide.html