Groupware Support for Software Requirements Inspection

M. Halling, P. Grünbacher, S. Biffl

***Abstract*—The inspection of software products has proven to be an effective approach to find defects. Inspecting requirements documents offers especially large benefits as it removes defects very early in the development process. On the other hand inspection is also an expensive and sometimes cumbersome process resulting in a large amount of inspection material that has to be sorted, searched, and consolidated. Existing inspection tools whose success has been empirically evaluated are focused on code inspection and fall short for inspection needs of early life cycle documents like requirements specifications. Based (a) on empirical data from our experiments with paper-based inspection of requirements documents and (b) on our experience with groupware support for software requirements negotiation, we have developed a concept for a groupware-supported requirements inspection process. In this paper we present our concept, discuss potential benefits for software requirements inspection, and propose an approach for empirical evaluation.**

***Index Terms*—Inspection, requirements, groupware, empirical evaluation criteria.**

I. Introduction

Software developers favor quality assurance approaches, which help them to determine and improve product quality effectively and efficiently. In the early stages of development, work products are usually less complex than during implementation and defects found have particularly high potential to save rework effort in later stages of development. Thus the inspection of requirements and design artifacts promises better leverage of inspection efforts compared to code inspection.

Major activities in the inspection process [20] are inspection planning, individual defect detection, defect collection in a team meeting, and inspection evaluation, followed by a rework based on the found defects:

The *planning* of the inspection process has to consider the potential effectiveness and efficiency of different inspection designs in a given project context. The inspection manager plans – according to the overall project plan – the products that should get inspected, the size of the inspection team, and the defect detection aids to be used. An important assumption for this planning is the effect an inspection process design will have on development, e.g., the delay caused by locking work products or development resources used for inspection.

Inspection evaluation determines the actual effectiveness of an inspection activity based on an estimate of the total number of defects (in a certain defect class) in the product under inspection. This evaluation is usually done after finishing all defect detection activities, but could also help to monitor individual steps during a defect detection process.

During *individual defect detection* each inspector uses a defect detection technique to look for specific classes of defects in a defined part of the inspected product [1]. This focus helps to make individual inspectors more effective than unfocused reading of an entire document. The defect lists produced by individual inspectors are input to the defect collection activity.

During *defect collection* the author, and sometimes the inspection manager, compile a common team defect list for rework. This activity is usually conducted as an inspection meeting. The meeting provides a forum for inspectors to discuss open issues, which could not be resolved individually. Furthermore, performance data on the inspection (e.g., effort, duration, number of defects found in certain defect classes) is gathered for the evaluation of the overall process.

Empirical studies on paper-based inspections [3][12][15][19] show that inspections are an effective approach to find defects in software products. But these studies also demonstrate that a purely paper-based approach is very expensive and critical needs of inspectors and inspection management are hard to meet with paper as the major means of communication:

- Typically a lot of paper, e.g., versions of the inspected documents, stacks of defect reports and inspection notes, has to be sorted, searched, and consolidated. This tends to be a time-consuming, error-prone, and tedious activity.
- As important management information is scattered across the paper documents, an in-depth evaluation can be made only after the inspection, which does not allow in-process inspection control.
- The effort involved in paper-based communication increases the cycle time of an inspection, which makes flexible approaches with several inspection cycles infeasible due to cost constraints.

In this paper we suggest to represent all documents used or created during an inspection in an electronic format, that explicitly represents the semantic structure of the document to be inspected, and allows automating some inspection ac-

Michael Halling is with the Institute for Software Technology at Vienna University of Technology, Karlsplatz 13, A-1040 Vienna, Austria and with the Systems Engineering & Automation department at Johannes Kepler University Linz, Altenbergerstr. 69, A-4040 Linz(e-mail: halling@swt.tuwien.ac.at).

Paul Grünbacher is with the Systems Engineering & Automation department at the Johannes Kepler University Linz, Altenbergerstr. 69, A-4040 Linz, Austria, (e-mail: pg@sea.uni-linz.ac.at).

Stefan Biffl is currently with the Fraunhofer Institute for Experimental Software Engineering, Sauerwiesen 6, D-67661 Kaiserlautern, Germany; on sabbatical leave from the Vienna University of Technology, Austria, (e-mail: Stefan.Biffl@tuwien.ac.at).

tivities. We expect the following benefits from tool support of inspection activities:

1) *Improved handling of shared documents.*

- We aim at reducing communication overhead through shared electronic documents instead of paper copies, which allows inspection management to get more accurate and timely feedback on the actual status of inspection tasks as well as faster feedback on product and inspection quality.

- Faster delegation and feedback cycles with more flexible collaboration processes will allow better support for inspection management.

- Electronic communication of defects and annotated documents, as well as support for both synchronous and asynchronous work will accelerate the inspection cycle.

2) *More efficient communication and defect collection in the inspection team.*

- From semantically structuring the inspected documents we expect less effort for defect collection when to decide on whether two defect descriptions refer to the same defect.

- We expect faster and more clear communication among the members of the inspection team and between team members, authors and the inspection manager.

- We also want to support joint meetings including synchronous work on the inspected documents.

3) *Cognitive support for individual defect detection.*

- Navigation along the semantic structure of the inspection document should provide better overview for the individual inspector during defect detection.

- In addition there is better handling of electronic document structure and content (i.e., annotating, searching, indexing, versioning).

Tool support for inspection has been empirically evaluated for environments automating the inspection of source code and showed similar effectiveness and efficiency as paper-based inspections (refer to Section II and [22]). Existing tools provide capabilities to represent electronic documents and typically allow annotations associated to a particular line of text. This association usually does not provide semantic information that would be important especially for early life-cycle artifacts like requirements. Furthermore, these systems focus on code inspections and fall short to support flexible support of inspection process variants.

Groupware tools have been used extensively to support collaborative processes in software engineering, e.g., requirements negotiation [5][13]. The inspection process itself is a collaborative process with a number of different roles (i.e., authors, inspectors, inspection manager, moderator). We thus propose a groupware concept supporting the inspection process and techniques discussed above. Our interest is not only the meeting activity itself, but especially individual defect detection and inspection management. We focus on supporting the individual defect detection activity with reading techniques, as this activity has been found to be particularly effective and provides direct input to the inspection team meeting activity [20].

The paper is structured as follows: Section II discusses related work on tools for inspection and current shortcomings dealing with early life-cycle documents and collaborative aspects of inspection. We propose possibilities for extension and alternative approaches. Section III provides the rationale, potential benefits, and evaluation criteria for groupware-support requirements inspection. Section IV introduces the major four activities of our groupware-supported requirements inspection process. Section V summarizes the concepts presented in the paper and provides an outlook on further work.

II. Related Work on Inspection Tools

In this section, we briefly present a set of evaluation criteria, describe the main characteristics of existing inspection tools, and outline issues that need to be addressed. Table A1 in the appendix summarizes features of the main existing tools based on information from a recent survey [21].

A. Inspection Management

The first group of evaluation criteria deals with *inspection management*, i.e., services needed throughout the entire inspection process. An inspection tool should achieve a high level of flexibility to tailor the process to a given context. So far most existing tools were specifically developed for one specific *inspection process*, e.g., SCRUTINY for the inspection method of Bull HN Information System [7]. ASSIST, the Asynchronous/Synchronous Software Inspection Support Tool [22], and CSRS, the Collaborative Software Review System [18], offer a process definition language to tailor the tool to any inspection process.

Another aspect of inspection management evaluates whether a tool supports asynchronous and/or synchronous inspections (i.e., *inspection meetings*). Tools that focus explicitly on asynchronous inspection processes, like WiP [16] and CSRS, do not offer any meeting services. Other tools, allowing for or focusing on synchronous inspections have at least some functionality required in same-time meetings (with the exception of ICICLE [2]). The latter group of tools usually also provides some sort of *decision support* for accelerating the team decision-making process (e.g., polls or voting).

The last criterion summarizes *inspection process evaluation* services. These services are based on automated data collection facilities (e.g., defects, time stamps) that are supported by all tools in our overview. Additionally all tools offer or at least indicate support for *a posteriori evaluation* of the performance using simple statistics and summaries.

However, *in-process evaluation* and in-process adjustment of the inspection process represent additional requirements for inspection tools. This means that inspection management should have the possibility to monitor and assess the performance of current inspection activities and to change certain process parameters to improve performance. This functionality is not offered by any of the surveyed tools, respectively is not explicitly mentioned in any documentation.

B. Document Handling

In addition to inspection management, *document handling*

is a key evaluation criterion. In this context the term "document handling" includes the types of documents supported by the tool, the representation of the information described in these documents, and the creation of linked annotations.

As far as document types are concerned all tools except ASSIST are limited to plain text documents. The main reason for this is that they were developed for source code inspections and there was no need to support different document types. ASSIST approaches this problem by providing the possibility to define any type of browser to view and inspect documents. The documents to be inspected are usually displayed in full text in the browser allowing the inspector to attach annotations to each line or to areas of multiple lines. Only CSRS transforms the document into a series of nodes allowing inspectors to make annotations for each node.

C. Defect Detection Support

Finally, we want to evaluate existing tools with respect to *defect detection support*, i.e., documents and services provided by the tool to facilitate defect detection. Most tools include functionality supporting the inspector with information from various *supplementary materials*, like checklists.

However, most tools do not explicitly consider documents for more sophisticated approaches like scenario-based reading techniques [20]. These techniques require additional supplementary material, like process description, high-level inspection goals, and thus make it further necessary to provide *information specific to context and inspector role*. Inspectors should only receive information that is relevant to them depending on their current position in the inspected document, their progress in the inspection process (i.e., tasks already fulfilled and tasks to be done) and their assigned inspector role. As the idea of different inspector perspectives coming from scenario-based reading techniques is rather new, none of the investigated tools explicitly supports this approach.

D. Empirical Evaluation

The presented tools have been empirically evaluated to varying degrees in concrete project situations. MacDonald and Miller [21] report that in most cases there was no comparison to a paper-based inspection process in the same environment. Therefore it is currently hard to assess whether and how much the existing tools improve the performance of inspections. However, most qualitative statements based on these empirical evaluations point out that there is evidence that tool-based inspection is not less effective than paper-based inspection – actually a rather unsatisfactory result.

An existing application of a Group Support System (GSS) for supporting the inspection meeting process [10][11] presents empirical evidence from field studies in professional environments that a GSS can significantly increase the performance and the overall contribution of inspection meetings. The GSS in this application is mainly used as an Electronic Meeting System. As our GSS-supported approach presented in Section IV focuses on individual defect detection and inspection process management, the results from [11] complement the concepts presented in this paper.

III. RATIONALE FOR GROUPWARE-SUPPORTED REQUIREMENTS INSPECTION

In this section we provide a motivation and argumentation for groupware-supported requirements inspection. Our experiences are based both on empirical data from experiments with paper-based inspection of requirements documents [3][15] and on experience with employing tool support for the negotiation of software requirements [14].

A. Paper-based Inspection of Requirements

Software inspection currently is the most effective way of checking early life-cycle documents in practice [12]. Software inspections represent a family of highly formalized processes that are designed to uncover defects in software development artifacts. While the potential benefits of inspecting early life-cycle documents are higher than for code inspections, the same is true for the costs [20].

In our paper-based experiments [3][15] we basically followed the traditional inspection process [8][12] comprised of the activities planning, individual preparation, meeting, defect removal, and evaluation. As mentioned before we focused on inspection of requirements documents. The requirements document we used for experimentation with paper-based inspection follows the Unified Process and uses UML [9] to formalize use-cases and domain object models. The outline of this requirements document is as follows: context information in plain text, functional requirements using use-case diagrams and descriptions, domain information via a domain object model and data descriptions in tabular form (see Fig. 2 in Section V as an example).

The goal of our paper-based studies was to evaluate the performance of individual defect detection techniques namely reading techniques. Reading techniques support inspectors in finding defects [1][19]. We distinguish checklist-based and scenario-based reading. Checklist-based reading techniques use catalogs describing potential defects in a very general way. Inspectors are not told what to do but rather what defect symptoms they could potentially find. In contrast, scenario-based reading techniques define a defect detection process that tells inspectors what to do and assigns certain scenarios to each inspector.

An example for scenario-based reading techniques is called perspective-based reading, which assigns one of three possible perspectives to an inspector: user, designer or tester. For example, in the case of the user perspective the inspector concentrates on verifying that the functional requirements are consistent and that they fulfill the user requirements of the target system. S/he extracts information from the requirements document, builds a use case model for abstraction, and checks the use case model for possible problems with the original requirements text.

Regnell *et al. [24]* summarize main empirical results of scenario-based reading performance and point out that these results are not clear. In some experiments scenario-based reading significantly outperforms checklist-based reading, in some experiments there is no significant difference.

In our two large-scale experiments that we conducted in an

academic environment [3][15] with team sizes varying between 5 and 6 inspectors, we found that scenario-based reading helps to direct inspector effort on certain parts of a document, and increases the detection of major defects.

For practical purposes, the benefits and costs of an inspection technique must be compared and evaluated in an economic context [4]. While inspection benefits can be quantified as saved rework effort from later phases, inspection costs are measured in working hours invested.

Based on our empirical studies, we identify the following two main arguments for developing a groupware-supported inspection process:

- *Reduction of cost drivers:* The two main cost drivers of inspection are the individual defect detection effort and the loss of inspection efficiency due to defect overlap (i.e., the number of defects found by more than one inspector) in an inspection team. While groupware supports reducing the effort by managing the documents and supporting the defect detection activities, it helps to diminish the overlap by providing different levels of inspector communication.
- *Control of Inspection Process:* Another important finding of our empirical studies is that inspection effort varies considerably among individual inspectors. As far as individual defect detection effort is concerned we observed values between 2 and 8 hours [3][15], but were not able to find intuitive inspector-specific parameters that explain these variations. These findings confirm that there is a considerable risk of inefficient inspection runs [12], which can be reduced through the application of the groupware-supported inspection process. Using groupware enables the inspection manager to monitor and, if necessary, adjust the task allocation in the inspection process in order to avoid clearly inefficient inspections.

B. Groupware Support for Collaborative Processes

Our discussion above showed that automating the a requirements inspection process is a challenging task:

- The process is made of activities with different characteristics (e.g., individual vs. concerted efforts, distributed vs. face-to-face meetings).
- The environment has to provide configurable access to shared deliverables, some of them structured in a complex manner.
- The tools have to support work techniques adopted in the methodology, e.g., reading techniques.
- The environment needs to provide views and perspectives for different users/roles in the process.

A groupware system for requirements inspection should thus seamlessly support the many ways that people work together; as individuals or in groups, co-located or geographically dispersed, synchronously or asynchronously.

Groupware is a breed of computer technology that has emerged in the last decade targeting team productivity and supporting groups of people engaged in a common task or goal. Groupware technology is used widely to communicate, cooperate, coordinate, solve problems, or to negotiate.

Within the vast number of groupware technologies Group Support Systems (GSS) focus on supporting group decision-making. A GSS is not just a single piece of software, but a collection of computer-based collaborative tools (e.g., for idea generation, idea organization, idea evaluation). A GSS typically includes software tools for brainstorming and idea generation, issue structuring and categorization, topic exploration, issue prioritization and voting, and logging. Although the tools appear simple to the users, their design derives from cognitive and social research findings and from extensive experience in the field with teams trying to accomplish meaningful work.

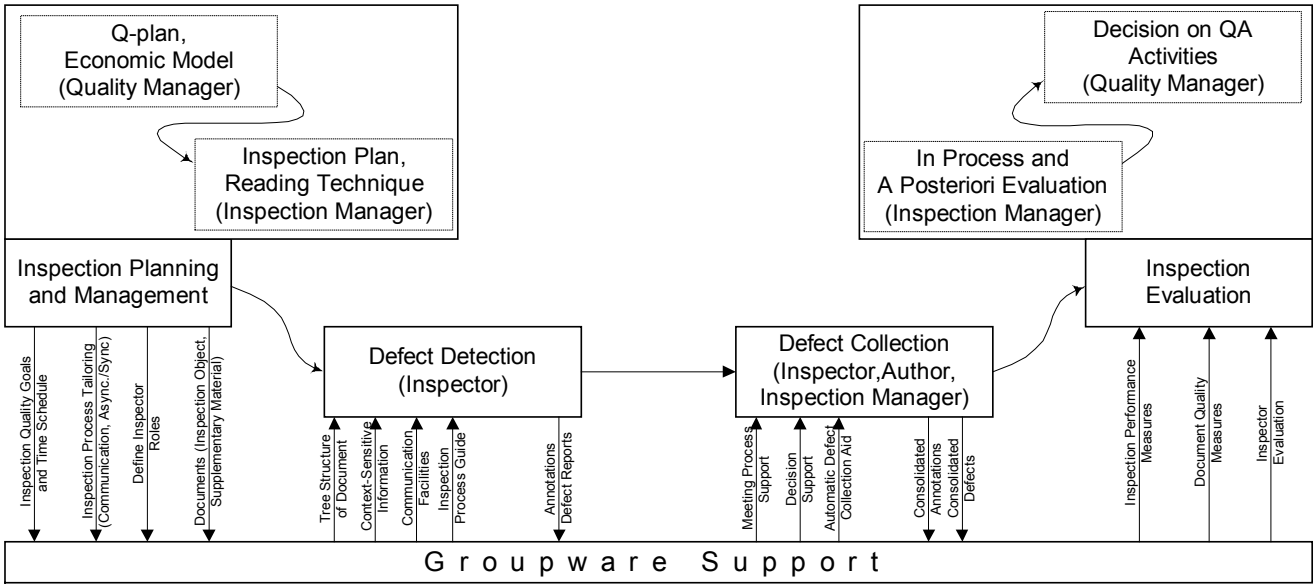
More than a decade of research has shows that under certain circumstances, teams using Group Support Systems (GSS) can save as much as 50% of their labor hours, and can cut their project cycle times by up to 90% [23][6]. Under the right conditions, teams using GSS can be far more productive than teams using pen-and-paper methods.

We have decided to use a Group Support System (GSS) to support our inspection process [3][15] for several reasons:

- A GSS provides a set of configurable groupware tools supporting different collaborative activities for the collection, categorization, and evaluation of artifacts.
- A GSS can be configured in a way that it supports collaborative activities as well as role-specific individual efforts in a shared workspace.
- A GSS supports the concept of a meeting leader and meeting participants with different, configurable access rights to ease meeting management.
- A GSS can be configured to visualize, edit, and annotate structured documents.
- The use of a GSS for collaborative software engineering has been successfully demonstrated in related areas of software engineering like code inspection [11] and software requirements negotiation [5][6].
- A GSS provides high-level groupware building blocks that allow rapid prototyping and delivery of collaborative methodologies, which allows early feedback of users.

Initially we want to support existing reading techniques that were originally designed for paper-based inspection [12]. In a next step the groupware-supported inspection process offers the potential of designing special reading techniques that are optimized for tool-based inspection.

Fig. 1: A Groupware-supported Inspection Process Framework



IV. Research Objectives and Evaluation Criteria

Based on information presented in the previous sections, drawbacks of existing tools and empirical evidence from experience with paper-based inspection, we want to address the following main goals:

1) *Improve inspection management with electronic inspection documents (inspection object, defect lists, feedback; reading techniques) and improved communication among team members.*

a. Inspection managers can implement different process designs (e.g., asynchronous/synchronous process, the amount of allowed and supported communication) and can tailor them to the project's needs.

b. On-line feedback and in-process monitoring of inspection activities becomes possible. Inspection managers can evaluate the performance of detection tasks in the current inspection run and can readjust the process in the case of severe problems.

c. Based on the data collection during each inspection run (e.g., defect lists, time stamps) the inspection manager can assess the inspection process quality, the probable document quality, and the performance of individual inspectors.

2) *Improve the handling of documents required during the inspection process (e.g., inspected software artifact, reading techniques, supplementary material).*

a. The groupware tool provides a structured representation of the inspection object according to meaningful entities in the document. These semantic entities (e.g., functional requirements, domain model objects, high-level goals) facilitate defect documentation, communication and matching.

b. The sharing of structured inspection results between all team inspection members (i.e., inspector, author, inspection management) is supported and easily possible.

c. The effort for defect, or more general annotation, collection is reduced. Similar defects are more easily identified due to the limited number of semantic entities of the inspection object.

3) *Support individual defect detection of each inspector.*

a. Context of the meaningful semantic structure of the inspection document aids fast navigation in the electronic version of the document.

b. If the reading technique asks the inspector to build new software model representations as part of inspection, an electronic tool can provide appropriate templates, which help to prevent double work during model building and possibly to reuse existing work.

c. The groupware tool uses available information on inspector role, current focus with the inspection object, and current step of the defect detection process to point the inspector to the subset of information that is needed in this situation. Therefore the groupware tool performs a filtering task releasing the inspector from the management of all the supplementary material available.

Based on these improvements, we expect the groupware-supported inspection's effectiveness (i.e., the number of defects detected) and efficiency (i.e., the number of defects found per invested hour) to perform better than the paper-based inspection process due to the following relationships:

1. We reduce the effort required to perform certain defect detection tasks, e.g., consistency checks between different documents or different parts of one document, creation of models. We further release the inspector from overhead effort like document management.

2. We reduce the overlap between individual inspectors by allowing and supporting different levels and ways of communication, e.g., each inspector's annotations and potential defects are 'publicly' observable.

We plan to conduct a controlled experiment in an academic environment to empirically evaluate the comparative advantage of our groupware-supported inspection process to

traditional paper-based inspection.

As outlined in [17] the difference between students and professionals might not be as significant as often argued in the empirical software engineering community. As we have experience with large controlled experiments in an academic environment, we will use them to get first indications on how tool-oriented and paper-oriented inspection relate to each other.

For quality and inspection management in professional environments we expect the groupware-supported inspection process to be an attractive option as it is based on a commercially available group support system and also offers more immediate inspection process control.

V. A Groupware-supported Software Inspection Process

In this section we present the groupware-supported inspection process in detail. Fig. 1 shows the groupware-supported inspection process framework. It shows the inspection process activities and how they communicate with the groupware tool, i.e., deliver input to and receive output from the groupware tool. In the following we discuss each process activity and describe in detail how the groupware tool supports it. We use the Group Support System (GSS) GroupSystems as a platform for our realization. GroupSystems software was developed at the University of Arizona and commercialized by GroupSystems.com.

A. Activity 1 – Inspection Planning and Management

The Inspection Planning and Management activity prepares the groupware tool for a concrete inspection run (see Fig. 1). The inspection leader/manager is responsible for a specific inspection run in general and in particular for this process activity.

The inspection leader must ensure that all *documents* required for the inspection (e.g., requirements document as inspection object, reading techniques, supplementary documents) are appropriately represented and configured in the GSS tool. For an optimal support of the individual preparation phase we suggest to extract the structure of the inspection object, e.g., the requirements outline from a requirements document (see Fig. 2).

The GroupSystems tool partly automates this step for text documents. Only information represented in figures and tables must be extracted either manually or semi-automatically (depending on the data format of the requirements document) and integrated in the document structure.

Note that there is no full text version of the requirements document within the GSS tool, rather the contents of the document is semantically grouped and organized in a hierarchy of concepts. In addition the electronic full text version of the specification document including tables and figures is available to the inspectors via external browsers.

The inspection leader must also link certain parts of the defect detection techniques to nodes in the requirements hierarchy. This is necessary to enable context-sensitive information for inspectors with defect detection aids and other supplementary material. S/he might for example determine that a certain set of questions of a checklist is only relevant for certain functional requirements, while another set is relevant for the domain model. If requirements documents and defect detection techniques have a similar structure across a company, this context-sensitive linking needs to be done only once.

Furthermore the inspection leader has the possibility to *tailor the inspection process.* The definition of the inspection process includes deciding between synchronous or asynchronous activities, determining the levels of communication support and defining the concrete process steps to be followed (e.g., synchronous inspection without meeting).

In order to enact the process definition, the inspection manager selects the participating *inspectors* and assigns them *different roles* and defect detection tasks with reading techniques. These reading techniques can be based on a checklist or on a set of scenarios. The inspection manager's decision on the concrete inspection process for a specific project situation also considers *quality goals and time schedule constraints* imposed by the project and/or quality management.

A main feature of the groupware-supported inspection process is to support any inspection process, which enables inspection managers to take specific project situations into consideration and implement the "optimal" inspection process for any particular situation (see research goal 1a in Section IV).

Furthermore this high degree of flexibility allows changing the inspection process during an actual inspection. This enables the inspection manager to quickly react to developments and experiences gained during a specific inspection run and to further optimize the inspection process (see research goal 1.b in Section IV).

B. Activity 2 – Individual Preparation

After the inspection-planning phase, the individual preparation phase (i.e., defect detection) follows according to the standard Fagan inspection process. A very important aspect of tool-supported individual preparation is functionality for document-handling. In this context the most important dimension is the definition of a defect location. Most existing inspection tools (see Section II and Table A1 in the appendix) provide full text in a browser and offer the possibility to assign a reported defect or an annotation to a line of code, some to single words and some to arbitrary lengths of text. Assigning a reported defect to a single line in a document might be reasonable for code documents but definitely unreasonable for a requirements document where the line alone usually has no meaning.

Therefore we view a requirements document as a structured document that can be represented as a hierarchy of concepts and requirements and visualized in a tree (see Fig. 2).

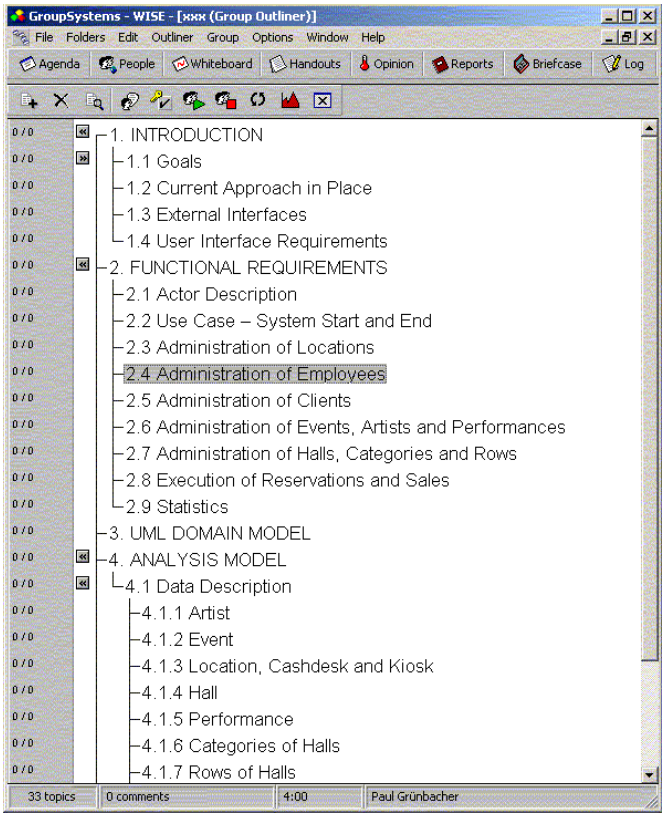


Fig. 2: Structure of a sample requirements document in the group support system (GSS) tool for outlining.

A reported defect can then be assigned to a specific node in the requirements tree. The main advantage of this approach is that each node contains context information and can therefore be basically analyzed without reference to other document parts. Another advantage of this approach is that there are a limited number of potential defect locations that simplify the comparison and collection of inspector annotations (refer to research goal 2 in Section IV).

For the individual preparation process we suggest that inspectors still use a paper-based version of the requirements document for reading. Harjumaa and Tervonen report empirical evidence that reading electronic documents is less effective than reading printed material [16]. However, if inspectors detect a potential defect, they can easily select the appropriate node of the requirements tree and make an annotation, i.e., a defect report or a comment, for this particular node in the requirements tree.

As one aspect of supporting the defect detection tasks, we suggest providing the electronic full text version of the requirements document including tables and figures in an external browser (a plain text version can be provided in the GSS tool). Thus allowing the inspector to automatically search for words and phrases in the document. This is actually a very simple but powerful support for the inspection of early life-cycle documents, like requirements documents, where related information may be spread over multiple sections and represented in different models. Suppose an inspector reads through the functional requirements and finds a term that is not clear in the current limited context. Now in a plain paper-based inspection there are two possibilities. He can either report the unclearness as a defect, which risks a false positive, or search through the remaining document to find a definition of the term, which is tedious and time-consuming. If there is an electronic version available, the inspector can simply search for the term and then determine whether to make an annotation.

By simply providing the electronic full-text version, we identify one way to provide the functionality to support individual defect detection (see research goal 4 in Section IV). However, another approach to achieve this goal is to provide context-sensitive information to the inspector and to actively lead the inspector through the defect detection process. With context-sensitive information we mean that depending on the current position of the inspector in the requirements document, he receives information on which checks to perform (checklist-based reading), on models to build (scenario-based reading) or just on his general inspection goal for this section (e.g., gather information or verify quality carefully). When the inspector has finished the task, he informs the system of the completion and then receives his next task.

The groupware support of the individual preparation phase further includes the provision of communication facilities that allow easy sharing of information among inspection team members. This communication possibly might increase individual inspection effectiveness and efficiency as inspectors receive in-process more detailed information on the quality of the inspection object and thus can focus their effort on those parts that require more attention. However, the inspection manager can tailor the amount of communication provided. There is a large spectrum of communication designs available ranging from no communication at all (like in the individual preparation phase of the standard paper-based process) up to full communication (every inspection participant receives on-line any information and all annotations of all other team members). An important aspect of our empirical evaluation work will focus on how different levels of communication influence the performance of individual defect detection.

C. *Activity 3 – Defect Collection/Team Meeting*

After the individual preparation phase the standard inspection process suggests conducting an inspection meeting to collect and filter defect reports and to briefly discuss unclear points and issues.

The aspect of defect collection is simplified in our approach as inspectors report annotations and defects linked to common semantic nodes in the tree that was generated from the inspection object (e.g., requirements tree). Therefore there is a limited number of defect locations where each location represents a certain semantic context. This is expected to facilitate the collection and filtering of annotations and defect reports. (see research goals 2b and 2c in Section IV).

In addition to collecting and filtering defect reports, inspection meetings should yield new defect reports due to the exchange and brief discussion of defects found during indi-

vidual preparation. Groupware tools can very effectively support such inspection meetings [10][11] by providing voting, discussion and decision support tools. However, a detailed description of these meeting support techniques is beyond the scope of this paper.

However, the electronic support of meeting processes might represent another key advantage of groupware-supported inspection as there is empirical evidence that meetings in paper-based inspections often show a poor performance [25]. Generally we suggest conducting an inspection meeting, if inspector feedback after the individual preparation phase indicates that serious quality issues exist.

D. Activity 4 –Inspection Evaluation

A key advantage of applying the groupware-supported inspection process is the immediate possibility to evaluate inspection performance, as a large variety of data is electronically available. As far as inspection process evaluation is concerned, we distinguish *in-process* and *a posteriori* evaluation (see research goal 1b and 1c in Section IV).

In-process evaluation means that the inspection leader receives certain inspection measures (e.g., list of tasks started/finished, effort used for certain tasks) on-line during an inspection run, e.g., the number of annotations; the effort spent on each defect detection activity. He can then use this information to take appropriate actions. For example he could decide to cancel certain defect detection activities, if no defect reports were registered after a certain amount of time. This fast feedback enables the inspection leader to appropriately adjust the inspection process in order to ensure that the inspection pays off in the end, e.g., reallocate tasks among inspectors to optimize overall performance in the team.

A posteriori data analysis aims at evaluating the costs and benefits of past inspection runs with the purpose of learning from them and of improving effectiveness and efficiency for further inspection runs. The most important evaluation aspects include:

- *Document Quality:* From the electronic defect lists and annotations the project manager can assess the quality of the inspected object. Furthermore it is possible to use different defect content estimation techniques to further estimate the defect potential of the document. This information can be used to plan additional inspection cycles and to adjust the project plan appropriately.
- *Inspection Process Quality:* Inspectors' compliance with the inspection process can be analyzed from their time-stamped data entries. Moreover, it is possible to assess the effectiveness of different defect detection activities.
- *Inspector Performance:* It is easy to assess the performance of each individual inspector. This supports inspection management in determining especially productive inspector-reading technique combinations.

Both in-process and *a posteriori* evaluation represent important foundations for empirical inspection process improvement. Inspection process monitoring and appropriate process tailoring to specific project situations are important features for increasing the acceptance of the inspection process in practice. Using the groupware-supported inspection process decreases on the one hand the risk of ineffective and inefficient inspections. On the other hand, it increases the probability of developing a highly beneficiary inspection process optimized for the specific project situation.

VI. Conclusion and Further Work

Although there is consensus that inspections are important, especially in early development stages, and there have been attempts to automate the inspection process, there still is a need for integrated support for the whole inspection process. These process activities with many collaborative elements are so far only dealt with partially and not in an integrated way by existing inspection tools.

In this paper we provided a concept to use a groupware support system to address the needs of requirements inspection. In contrast to existing inspection tools, GSS technology fits ideally for this purpose because it provides a flexible and powerful set of tools to support the entire inspection process. Main features and benefits of the resulting groupware supported requirements inspection process are:

1. Improved inspection management with electronic representation of inspection documents (inspection object, defect lists, feedback; reading techniques).
2. Improved handling of inspection documents using a semantic structure of the inspected document for context and navigation.
3. Tool support for reading techniques in the individual defect detection activity.

Further work will be to evaluate the effort and potential problems incurred when customizing a COTS GSS for the groupware supported requirements inspection process. So far, our tailoring experience shows that there is some initial effort to tailor the GSS tool to the inspection process, which is clearly less than implementing a tool from scratch. Further, we estimate that the actual effort to prepare the previously tailored GSS tool for an inspection run in a specific project context will not exceed the effort to prepare a paper-based inspection.

Based on the evaluation criteria presented in this paper we will first replicate paper-based inspection experiments and compare the results of teams, who follow the traditional paper-based way, to teams, who use the groupware tool.

Based on these experiences we will in a second step develop alternative reading techniques, which explicitly exploit the advantages of the groupware system, and compare their performance, to teams, who apply the current best inspection practice with paper as medium. We will compare their effectiveness and efficiency, and further the effort needed for inspection preparation and the overall cycle time of inspection time.

For inspection management we will investigate the new means for timely evaluation of inspection, which are possible, if groupware is used. Result of the investigation will be an empirical contribution to which inspection tools work best under certain project conditions.

ACKNOWLEDGEMENTS

This work is based on the contributions of many people: The students and faculty involved in two paper-based inspection experiments at the Vienna University of Technology, with 200 persons involved in each experiment. Michael Halling has been supported by the Austrian Science Fund, Grant P-14128-COSIMIS. Stefan Biffl has been supported in part under the Austrian Science Fund, Grant J-1948-INF.

REFERENCES

[1] Basili V., Green S., Laitenberger O., Lanubile F., Shull F., Soerumgaard S., and Zelkowitz M., "The Empirical Investigation of Perspective-Based Reading", *Empirical Software Engineering: An International Journal*, vol. 1, no. 2, 1996, pp. 133-164.

[2] Bell Communications Research, *"ICICLE User's Guide"*, Bell, 1993.

[3] Biffl St., Halling M. "Software Product Improvement with Inspection", *Proc. Of Euromicro 2000 Workshop on Software Product and Process Improvement, Maastricht,* IEEE Comp. Soc. Press, Sept. 2000.

[4] Biffl St., Halling M., "A Framework for Economic Planning and Evaluation of Software Inspection Processes", *Proc. of the Workshop on Inspection in Software Engineering (WISE'01)*, July 2001.

[5] Boehm B., Grünbacher P., Briggs B., "Developing Groupware for Requirements Negotiation: Lessons Learned", *IEEE Software, May/June 2001*, pp. 46-55.

[6] Briggs R.O., de Vreede G.-J., and Nunamaker J.F., Jr., "Thinklets: Achieving Predictable Repeatable Patterns of Group Interaction with Group Support Systems (GSS)", *Proc. HICSS 2001 (Hawaii Int'l Conf. System Sciences), IEEE CS Press*, Los Alamitos, Calif., 2001.

[7] Bull H. N., *"Scrutiny User's Guide",* Information Systems, Inc., U.S. Applied Research Laboratory, 1994.

[8] Fagan M., "Design and Code Inspections To Reduce Errors In Program Development", *IBM Systems J.*, vol. 15, no. 3, 1976, pp. 182-211.

[9] Fowler M. and Kendall S., *"UML Distilled. Applying the Standard Object Modeling Language"*, Reading, MA: Addison Wesley Longman, 1998.

[10] Genuchten M., Cornelissen W., and Dijk C., "Supporting Inspections With an Electronic Meeting System", *J. MIS, vol. 14, no. 3, 1998,* pp 165-178.

[11] Genuchten M., Dijk C., Scholten H., and Vogel D., "Industrial Experience in Using Group Support Systems for Software Inspections," *IEEE Software, vol. 18, no. 3, May/June 2001*, pp. 60–65.

[12] Gilb T., Graham D., "*Software Inspection"*, Addison-Wesley, 1993.

[13] Grünbacher P., "Integrating Groupware and CASE Capabilities For Improved Stakeholder Involvement in Collaborative Requirements Engineering", *Proc. Euromicro 2000*, pp. 232–239, IEEE Computer Society. ISBN 0–7695–0780–8

[14] Grünbacher P., Briggs B., "Surfacing Tacit Knowledge in Requirements Negotiation: Experiences using EasyWinWin", *Proc. Hawaii International Conference on System Sciences*, IEEE Computer Society, 2001.

[15] Halling M., Biffl St., (2001) "Using Reading Techniques to Focus Inspection Performance", *Proc. of Euromicro 2001 Workshop on Software Product and Process Improvement, Warsaw,* IEEE Comp. Soc. Press, Sept. 2001.

[16] Harjumaa L. and Tervonen I., "A WWW-based tool for software inspection", *Proc. of HICSS-98, vol. 3*, 1998.

[17] Höst M., Regnell B., and Wohlin C., "Using Students as Subjects – A Comparative Study of Students and Professionals in Lead-Time Impact Assessment", *Empricial Software Engineering, 5*, p. 201-214, 2000.

[18] Johnson P.M., "An instrumented approach to improving software quality through formal technical review", *Proc. of the 16th International Conference on Software Engineering*, 1994.

[19] Laitenberger, O., *"Cost-effective Detection of Software Defects through Perspective-based Inspections", PhD thesis*, University of Kaiserslautern, Germany, www.iese.fhg.de, May 2000.

[20] Laitenberger O., and DeBaud J.-M., "An encompassing life cycle centric survey of software inspection", *Journal of Systems and Software*, vol. 50, no. 1, 2000, pp. 5-31.

[21] MacDonald F., Miller J., "A Comparison of Computer Support Systems for Software Inspection", *Automated Software Engineering 6*, 291-313, 1999.

[22] MacDonald F., "ASSIST V2.1 User Manual", *Technical Report EfoCS-28-98, Department of Computer Science, University of Strathclyde*, 1998.

[23] Nunamaker J., Briggs R., Mittleman D., Vogel D., and Balthazard P., "Lessons from a Dozen Years of Group Support Systems Research: A Discussion of Lab and Field Findings", *Journal of Management Information Systems, Winter 1996-97, 13(3),* pp.163-207.

[24] Regnell B., Runeson P., and Thelin T., "Are the perspectives really different? – Further experimentation on scenario-based reading of requirements", *Empirical Software Engineering, 5*, page 331-356, 2000.

[25] Votta L., "Does every Inspection need a Meeting?", *ACM Software Eng. Notes*, vol. 18, no. 5, 1993, pp. 107-114.

APPENDIX

TABLE A1:
OVERVIEW ON THE PERFORMANCE FEATURES OF TOOLS FOR INSPECTION SUPPORT.

Functionality	*GSS Inspection*	*ASSIST [22]*	*ICICLE [2]*	*Scrutiny [7]*	*WiP [16]*	*CSRS [18]*	*CSI / CAIS / AISA*
Inspection Management							
Inspection Process	Asynchronous and synchronous; tailorable to any inspection process	Asynchronous and Synchronous; tailorable to any inspection process	Synchronous; Standard Fagan Inspection	Synchronous; Inspection Method of Bull HN Information System	Asynchronous; GRCM quality model.	Asynchronous; tailorable to any inspection process	Synchronous; Asynchronous; Humphrey Model of Inspection
In-Process Inspection Management	Supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported
Meetings	Support for electronic meetings is a key competence of GSS.	Supported	Not supported	Supported	Not supported	Not supported	Supported
Decision Support (Polls, Voting)	Supported	Supported	Not supported	Supported	Not supported	Supported	Supported
Defect/Data Collection	Supported	Supported	Supported	Supported	Supported	Supported	Supported
A Posteriori Inspection Evaluation	Supported	Partially supported.	Partially supported.	Supported	Partially supported.	Supported	Partially supported.
Document Handling							
Document Types	Any type supported.	Any type supported.	Plain Text	Plain Text	Plain Text	Plain Text	Plain Text and Graphics
Document Representation	In the GSS the document and its information is represented in a tree; Full text available using external browsers.	Full text of document within an appropriate browser.	Full text of document within an appropriate browser.	Full text of document within an appropriate browser.	Full text of document within an appropriate browser.	Document is represented as a series of nodes with full text.	Full text of document within an appropriate browser.
Linked Annotations	Annotations of various types can be made for each tree node.	Line-oriented	Line-oriented	Line-oriented, Current Focus	Line-oriented	Annotations for each node.	Line-oriented
Defect Detection Support							
Supplementary Documents	Checklists, scenarios, task descriptions and further information is supported.	Supported; Checklists	Supported, but no checklists.	Not supported	Supported; Checklists	Not supported	Supported; Checklists
Context- and role-specific tasks/information	Supported	Not supported.	Not supported.	Not supported.	Not supported.	Not supported.	Not supported.