The systematic Adaptation of Perspective-based Inspections to Software Development Projects

Oliver Laitenberger, Kirstin Kohler

***Abstract*—** Software inspection is an established approach for detecting and removing defects immediately after software documents are created. However, the advance of software technologies, processes, and methods, such as the widespread adoption of object-orientation, raises new problems regarding software quality assurance with inspections. An important one is the question of how developers can perform the individual defect detection activity of an inspection in a systematic manner. Recent results suggest the use of scenario-based reading techniques for this purpose. Among the existing scenario-based techniques the perspective-based reading technique is particularly promising, since it can be used in a cost-effective manner for the systematic defect detection in various document types. Cost-effectiveness benefits, however, are only feasible if the technique is appropriately tailored to the characteristics of the software development project. The goal of this paper is to present a systematic process on how to perform the tailoring. An example is detailed to illustrate the process. Following the process described in this paper allows practitioners to upgrade their existing inspection approach with a more systematic reading technique, that is, to implement and perform perspective-based inspections. The benefits are more rigorous inspections with an improvement of inspection cost-effectiveness.

***Index Terms*—**Software Inspection, Reading Technologies, Perspective-based Reading

1. INTRODUCTION

Software inspection[1] is an approach that allows the detection and removal of defects immediately after software documents are created. Since the seminal introduction of the generic notion of inspection to the software domain in the early 1970s [8], it has evolved into one of the most cost-effective methods for early defect detection and removal. Its proponents claim that inspections can lead to the detection and correction of anywhere between 50 and 90 percent of defects [6] [10]. Moreover, rework costs can be reduced considerably, since defects are typically found directly after they were introduced. In addition to these quantitative benefits, inspections also expose qualitative ones. An example is the fact that early defect detection and removal improve the predictability of software projects and help project managers stay within schedule, since problems are unveiled throughout the early development phases. Costly rework cycles at the end of the development or maintenance project are therefore avoided. Another example is the learning effect of inspection participants that helps developers prevent defects in subsequent development phases. In this way, inspections help turning the defect detection into a defect prevention process [10].

A software inspection involves activities in which qualified personnel determine whether software documents are of sufficient quality for subsequent development activities. It usually consists of several steps including planning, defect detection, defect collection, and defect correction. An inspection organizer is responsible for planning. The defect detection and defect collection activities can be performed either by inspectors (i.e., developers) individually, or in a group meeting. Recent empirical findings reveal that the synergy effect of inspection meetings is rather low in terms of defects detected detected [12] [22] [26]. Therefore, defect detection should be considered an individual rather than a group activity. Defect collection, on the other hand, is often performed in a team meeting (i.e., an inspection meeting). The main goals of the team meeting are to agree on anomalies that inspectors have detected individually, to eliminate false positives, and to specify the defects for correction. An inspection usually ends with the correction of the documented defects by the author. Figure 1 illustrates the various phases. More information about the process, products, and roles of an inspection can be found in [14].

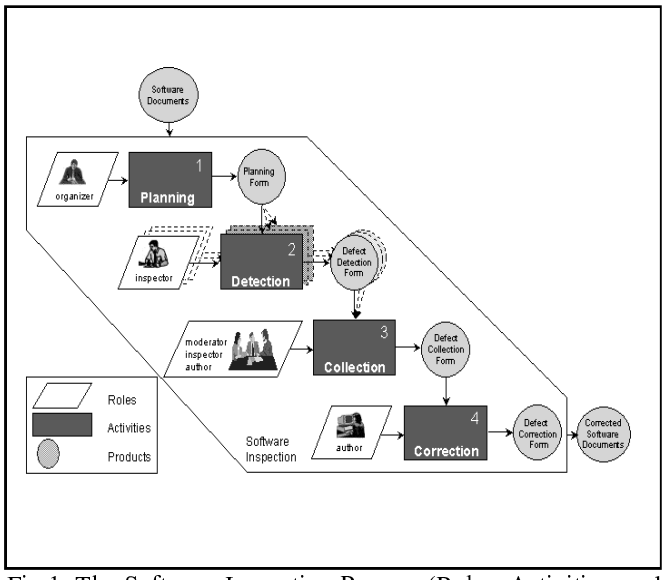

Fig.1. The Software Inspection Process (Roles, Activities, and Products)

• O. Laitenberger and K. Kohler are with the Fraunhofer Institute for Experimental Software Engineering, Sauerwiesen 6, D-67661 Kaiserslautern, Germany
E-mail: {Oliver.Laitenberger, Kirstin.Kohler}@iese.fhg.de

[1] In this paper, we use "software inspection" as an umbrella term for non-execution based quality assurance procedures. It includes, for example, Formal Technical Reviews (FTR), but excludes management or team meetings.

In existing inspection implementations the inspection results in the form of defects detected highly depend on human factors, such as the experience of inspectors and context criteria, such as the amount of preparation effort available. Both, the human dependency and the costs associated with inspections are primary reasons of practitioners for not adopting inspection technology on a larger scale. The many benefits of inspections, such as early defect detection and defect cost reduction, are often not considered in this discussion. Even if these benefits are taken into account, practitioners often argue that only minor defects, such as spelling mistakes, are detected. Major ones that have a significant impact on the quality of the system are only detected in later development phases, testing, or system usage. And this can really be the case if either very junior people will be involved in the inspection for learning purposes or if there is a shift in the development technology (e.g., moving to object-orientation). Since in both cases developers are unfamiliar with the artifacts, typical sources of defects, standards, guidelines, and other important context criteria, they do not know what to look for and how to perform the required quality checks in an inspection. Without any technical guidance or support, they are therefore not in the position to detect defects beyond trivial ones. But even for experienced developers, the inspection results are sometimes questionable for two reasons. First without any technical support, each inspection participant may look for the same defects or check the same quality attributes. Other, equally important quality attributes may get too little attention. As a consequence defects slip through the inspection process and propagate to later phases where their detection and correction costs escalate. Second, today's software artifacts can be very large. Without any guidance on what to check, most of the inspectors often perform their scrutiny sequentially. They start their checking activity at the beginning of document and read through the document page after page. However, because of fatigue and boredom effects the pages at the beginning of the artifact are getting much more inspector attention than the pages at the end of it. These effects are illustrated in the following figure that characterizes the attention level as a function of the size of the artifact inspected:

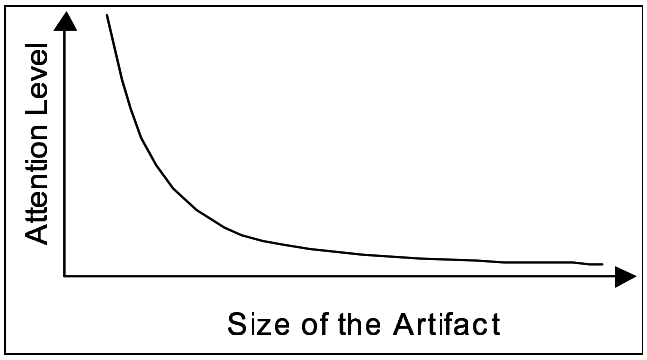

Fig.2. The Relationship between Attention Level and the Size of the Artifact inspected

According to Figure 2, all inspectors spend proportionally more effort for checking quality properties at the beginning of the artifact than at the end of it. As a consequence, the probability that a defect at or near the end of the artifact is actually detected is lower than for a defect at the beginning of the artifact. Hence, a better strategy would be to assure that all information in the artifact gets about the same level of attention from inspectors (i.e., that the effort density is about the same for almost every piece of information in the document).

To address most of these problems, Victor Basili proposed scenario-based reading [5]. The basic idea of a scenario-based reading technique is the use of so-called scenarios that allow inspectors to share the defect detection workload and at the same time offer more procedural support for inspectors (to alleviate the impact of human factors on inspection results). In the context of this work, a reading scenario is defined as an algorithmic guideline for the inspector that describes how to go about finding the required information in a software artifact, as well as what that information should look like. A scenario can therefore be regarded as the vehicle for individual defect detection in inspections.

Several examples for scenario-based reading techniques have been suggested so far [4][23][25]. Among them, perspective-based reading (PBR) is particularly promising [4][13]. The idea behind this technique is to let inspectors read the software document(s) from particular stakeholder viewpoints. For each of these viewpoints either one or multiple scenarios are defined. Each scenario provides a specific focus onto the artifact and the inspectors follow the scenario throughout their scrutiny. The success of the PBR approach has already been demonstrated in number of different empirical studies [4][23][25]. These studies have also showed that tailoring of PBR to project specifics and context characteristics is required to be most cost-effective. However, little information is currently available for practitioners on how to tailor it. This gap needs to be closed to broaden the applicability of the technique and to ensure its scalability to different project situations.

An initial step in this direction has already been described in [17]. However, we believe that the description is not detailed enough for practitioners to be directly applicable. This paper therefore details a full flavoured process for the systematic adaptation of the perspective-based reading techniques to the particuliarities of a software project after introducing the basic principles of scenario-based reading techniques and PBR. The process helps practitioners develop PBR scenario with which they can update their existing inspection implementation, that is, to implement perspective-based inspections. The process therefore ensures a smooth transition to a more rigorous scrutiny in an inspection with its positive impact on inspection cost-effectiveness.

The paper is structured as follows: Chapter 2 describes the essential elements of scenario-based reading techniques. Chapter 3 presents details of perspective-based reading.

Chapter 4 illustrates the process for the development and improvement of perspective-based reading scenarios. Chapter 5 finally concludes.

2. SCENARIO-BASED READING TECHNIQUES

2.1. Basic Idea behind Scenario-based Reading

Although reading is fundamentally about recognizing and understanding words, sentences, and graphics, people read in different ways and for different reasons. To understand different kinds of reading, it is useful to characterize reading along two dimensions: the nature of engagement with a document and the breadth of the activity across documents. The educator Mortimer J. Adler [2] describes the engagement with a document as varying from active to passive. Active reading combines reading with critical thinking, learning, and decision making, whereas passive reading is less careful and requires less effort. Active reading tends to involve writing, especially note-taking and annotation, while this is often not the case for passive reading. The second dimension - breadth across documents- varies with each reader and each type of reading. Reading a single document involves bookmarking and navigating, whereas reading multiple documents involves sorting, filing, and navigating. In the software domain, most existing reading approaches or techniques, such as ad-hoc (i.e., no reading support) or checklist-based reading [10] (i.e., a set of questions to answer throughout or after the scrutiny) , can be characterized as passive reading with a focus on at least two documents.

Active reading of multiple documents, such as suggested by Parnas and Weiss [20] [21], is important for the defect detection activity in software inspections. In our opinion, active reading of multiple documents is the vehicle for understanding information about a software entity. In cognitive science, understanding is often characterized as the construction of a mental model that represents the objects and semantic relations in a text. Hence, cost-effective reading techniques for defect detection in inspection must assist inspectors in the construction of their mental models by strengthening factors that support this process and by weakening those that impede it.

Two factors, in particular, are crucial in this respect: coherence as a positive influence on understanding and cognitive overhead as a negative one. A document is coherent if an inspector can construct a mental model from it that corresponds to facts and relations in a possible world. Cognitive overhead, on the other hand, is information that is unnecessary for understanding. A reading technique for the purpose of defect detection should therefore support the construction of a mental model and, at the same time, avoid cognitive overhead of inspectors.

The mechanism to achieve this goal is to focus the attention of inspectors to specific information in the document that is crucial for their understanding. The scenario-based reading techniques take advantage of this mechanism and, thus, influence both of these factors in the desired direction. A reading scenario supports the construction of a mental model by actively guiding people to construct artifacts while reading and, at the same time, avoids cognitive overhead by pointing inspectors to the relevant, limited set of information. In this way an inspector gets guidance in the construction of his or her mental model. The problem of cognitive overhead is alleviated, since the inspector does not have to understand each and every detail in the document. The concrete implementation of this idea becomes transparent after explaining the structure of the reading scenarios.

2.2. Structure of a Reading Scenario

To ensure that each inspector is following a specific reading approach, the scenario-based techniques provide guidance for an inspector in the form of scenarios [5][13][25]. A scenario explains how to read and examine the information in the documentation. In a sense, it provides algorithmic guidelines on how inspectors ought to proceed while reading. The guidelines in the scenario include procedures for extracting the information as well as procedures for examining the extracted information. The scenario itself consists of three major sections: introduction, instructions, and questions (Figure 3).

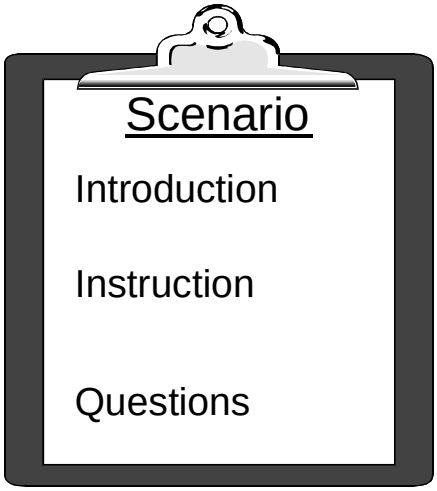

Fig.3. Generic Structure of a Reading Scenario for Perspective-based Reading

- The introduction part describes the quality attributes most relevant for the scrutiny of the inspector. It also includes a short description of the role or perspective taken by the inspector throughout the defect detection activity.
- The instructions describe activities about what kind of documents an inspector is to use, how to read the documents, and how to extract and examine the appropriate information from them. While identifying, reading, and extracting information, inspectors may already detect some defects. However, an inspector is to follow the instructions for three reasons. First, instructions help an inspector decompose large documents into smaller

parts. This is crucial because people cannot easily understand large documents. Understanding involves the assignment of meaning to a particular document or parts of it. It is a necessary prerequisite for detecting major defects, which are often the expensive ones to remove if detected in later development phases. Second, the instructions require an inspector to actively work with the documents. This ensures that an inspector is well prepared for the following inspection activities, such as the inspection meeting. Finally, the instructions help an inspector focus his or her attention on the information that is relevant for one particular stakeholder. This avoids swamping inspectors with unnecessary details and alleviates the problem with cognitive overhead.

- Questions at the end of the scenario ask if the examined information fulfills a set of given quality factors. Once an inspector has achieved an understanding, he or she can examine and judge on a solid basis whether the information as described fulfils the required quality factors. For making this judgement, a set of questions focuses the attention of an inspector on specific aspects of the information, which can be competently answered because of the understanding attained. In this way, the instructions put the inspectors in the position to make informed decisions while answering the questions.

3. The Perspective-based Reading Technique

Perspective-based reading represents an specific instantiation of the scenario-based reading techniques. The main idea behind the perspective-based reading technique is that a software product should be inspected from the perspective of different stakeholders [4] [7] [13]. The rationale is that there is no single monolithic definition of software quality, and little general agreement about how to define any of the key quality properties, such as correctness, maintainability, or testability. Therefore, inspectors of an inspection team have to check software quality as well as the software quality factors of a software artifact from different perspectives. The perspectives mainly depend upon the roles people have within the software development or maintenance process.

Figure 4 illustrates the idea of reading a document from multiple perspectives. There, a given document is read from the perspective of a designer, tester, and user.

For each perspective, either one or multiple scenarios are defined, consisting of repeatable activities an inspector has to perform, and questions an inspector has to answer. The activities are typical for the role within the software development or maintenance process, and help an inspector increase his or her understanding of the software document from the particular perspective.

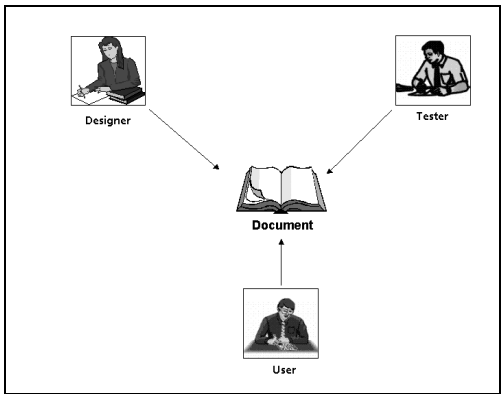

Fig.4. Reading a Document from Multiple Perspectives

For example, designing test cases is a typical activity performed by a tester. Therefore, an inspector reading from the perspective of a tester may have to think about designing test cases to gain an understanding of the software product from the tester's point of view. Instructions guide the scrutiny of the document. Once understanding is achieved, questions about an activity or questions about the result of an activity can help an inspector identify defects. An example of a perspective-based scenario is presented later on.

Reading a document from different perspectives is not a completely new idea. It was seeded in early articles on software inspection, but never worked out in detail. Fagan [8] reports that the real tester should inspect the piece of code. Fowler [9] suggests that each inspection participant should take a particular point of view when examining the work product. Graden et al. [11] state that each inspector must denote the perspective (customer, requirements, design, test, maintenance) by which they have evaluated the deliverable. However, if the stakeholders are available for the inspection, but inexperienced, they do not know what to look for and how to perform the scrutiny. In this situation, the PBR technique prevents defect slippage to later phases by providing concrete reading guidance for the specific stakeholders.

So far, the perspective-based reading technique has been applied to inspecting requirements documents [4], object-oriented design models [17], and code documents [15][16]. Moreover, it has been tailored to the Rational Unified Process [18] and component-based development [3]. However, there is no detailed description of how to set up the various scenarios. Practitioners, however, require this kind of information, since the full advantage of PBR can only be observed if the technique is adequately tailored to the project specifics.

4. A Process for PBR Scenario Development

The success of a scenario-based reading technique, such as perspective-based reading, relies on the ability of software engineers not only to follow existing scenarios, but to create new ones that fit in the development environment. In addition, the creation of new scenarios may be necessary because of the need to accommodate new types of documents, new defect categories, new stakeholders, or a new development paradigm.

Because of the fact that most of these characteristics are specific to development projects, the scenarios often have to be written from scratch before they can be reused in many projects of an organization. This requires activities (i.e., a process) that can be easily followed and results in the required scenarios.

4.1. Process Description

The process consists of several vital steps to come up with an initial set of PBR scenarios. Part of the process is the use of available stakeholder expertise and human capital in the specific project or organization. Hence, in addition to the description of each step, we provide questions the scenario developer might use for interviewing the various stakeholders to get the crucial information for scenario creation. In some cases we also give guidelines for the evaluating the results of the activities based on our experiences.

Figure 5 outlines the process for scenario development.

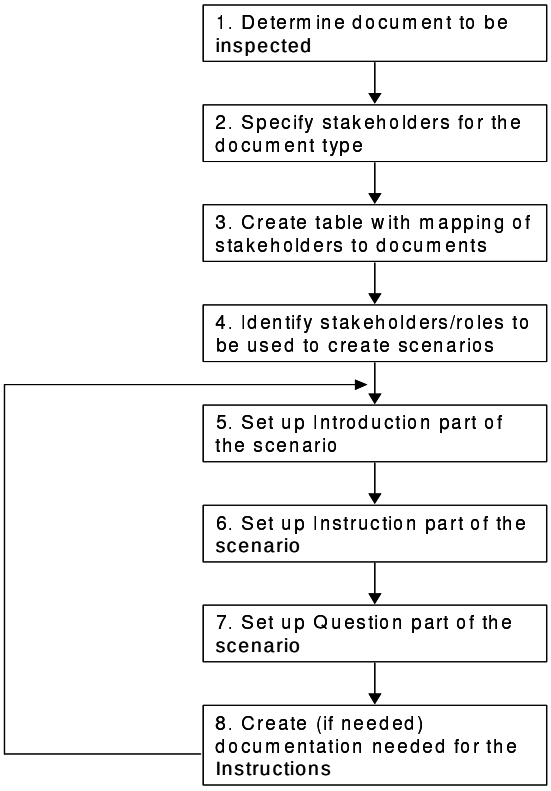

Fig.5. The Scenario Development Process

1.) Determine document to be inspected
In the process of scenario creation, the initial step is to determine the kind of information that needs to be inspected and the kind of documentation that is currently available for the information. An answer to these questions largely depends on the nature of the underlying development method. A product model of the development process might support the definition of an appropriate set of information and documentation to be inspected. In function-oriented development methods, the information typically relate to systems, subsystems, modules, or functions. The documentation typically involve requirements, design, or code documents. In object-oriented development approaches, information relates to classes, objects, and operations (i.e., methods) as well as systems, subsystems, and modules. In this case the documentation typically involve a set of (graphical) analysis and design models together with their textual description.

2.) Specify stakeholders for the document type
The different stakeholders that have a vested interest in the information under inspection need to be specified. As a starting point, the scenario developer may look at stakeholders that have a particular role in the software development process. Each role that contributes to the creation of the document, or uses (reads or changes) it as part of his/her work might be used as role for one scenario. For example, assume the document to be inspected is a requirements document. Throughout the development process the requirements engineer creates this document (i.e., documents the information on the expected functionality of a software system), the tester uses the information to generate test cases or run tests, the designer reads it to create the design, and the customer verifies whether the document describes all desired functionality. In this example there are at least four stakeholder roles, which might serve as an input for the creation of reading scenarios. Each of these roles has its own requirements in terms of quality and, therefore, has a different view on the document type and information. If a particular document is not of interest to any stakeholder, its value to the overall software development process is questionable. Helpful questions to identify all perspectives/roles are the following:
- Who reads this document?
- Who writes (part of) this document?
- Who decides upon the content of the document?
- Who has a special interest in the document?

3.) Create table with mapping of stakeholders to documents
The scenario developer identifies which of the roles need information from which document. In doing so, the scenario developer determines the information content of the document and identifies which part of the document and what kind of information in the document is most important for a particular stakeholder (e.g., to perform his or her role in the software development process). In many environments, the various document types, their contents and stakeholders' roles are included in the description of the software development process (i.e., the process documentation). The relationship

between document types and stakeholder roles can be best characterized and captured in a tabular format.

4.) Identify stakeholders/roles to be used to create scenarios

The scenario developer has to decide which of the identified roles should be used to create scenarios. The table resulting from step 3 may help identify the set of perspectives that provides full coverage of the information. It might make sense to exclude roles that have a very similar view on the document (such as a black-box and a white-box tester).

Each of the scenarios to be defined should have a title that specifies the type of documentation to be inspected and the name of the role the scenario is based on. The name of the role should reflect a role name that is well known in the environment. The names are often included in the process model.

5.) Set up introduction part of the scenario

For each of the scenarios the introductory part needs to be described. The scenario developer should name the role title for the chosen perspective. In addition the main goal and tasks of a person with that role have to be stated.

Helpful questions to define and describe the introduction are:

- What is the main task of a person having that role?
- What is the main interest of that role?
- What is the measure of success?
- When has she/he done a good job?
- Which quality aspect is this person most interested in?

The introduction should not be longer than 2-3 sentences.

6.) Set up instruction part of the scenario

In the next step the instruction part of the scenario has to be set up. The scenario developer describes the activities the stakeholder usually performs with the information in the documentation. The activities have to be phrased in a way that makes inspectors document the extracted and examined information (e.g., marking them with a colored pen or writing parts of the information down) and thereby force the instructor to actively work with the document under inspection.

Helpful questions are the following:

- What are the parts of the document the stakeholder is interested in?
- How does he/she find this information?
- In what section, picture, diagram, … can he/she find the information?
- What tasks do the person perform with that information?
- How does he/she perform that task?
- What documentation is he/she creating as part of that task?

The instruction part should not be longer than a page.

7.) Set up question part of the scenario

The next step in defining a scenario is to set up the questions an inspector is to answer based on the extracted information and the understanding of the information he or she has achieved. Characteristics of typical problems in a particular environment, exemplified by defect distributions, are useful information for developing the questions, since they are often typical representatives of problems in an environment. However, only those questions are to be included in a scenario that an inspector can answer with the understanding he or she can achieve based on the extracted information. While closed questions (i.e., questions with a yes/no answers should be used to check specific scenario results or quality aspects, open questions should trigger further defect scrutiny. The question part should not contain more than 7+/-2 questions [19].

8.) Create (if needed) documentation needed for the instructions

Finally all documents needed in order to perform the activities specified in the instruction part have to be created. This might for example include use case templates if the inspector has to create use-cases. These documents will be given to the inspectors together with the scenarios.

The granularity of the instruction part should have enough detail for an inspector to follow the given instructions in a step-by-step manner. This captures the information and quality attributes the inspectors have checked. If defects slip through the inspection process, this information can be analyzed to improve the scenarios. Hence, it becomes transparent how the inspectors achieved the results of the defect detection activity.

Both, the level of granularity and specificity depend on the knowledge of the inspectors. Junior inspectors require more precise and detailed instructions on how to perform the required checks than senior ones. For the latter the question part of a scenario may be more important, since they are already familiar with the activities of a specific perspective. For some perspectives it might be difficult to describe activities as part of a scenario that are small enough to be fully completed within the scope of the inspection. For example an inspector reading a requirements document from the perspective of an architect cannot define the entire architecture in its full detail. Hence, an adequate level of abstraction for the activity has to be chosen. In the example presented later on, the architect should not develop the entire architecture, but identify the main components of the system.

The instruction part of the scenario also allows the integration of more formal approaches for checking the

quality of information described in a document. The scenario-based approach therefore allows a smooth adaptation of formal techniques in inspection practice.
In different companies, we experienced very different capabilities of people to create scenarios. According to their maturity, scenario development required from half an hour to one day of effort. In general we observed, that it is much easier to create scenarios if the process of an organization is well defined with all responsibilities, roles, activities and documents. In a way, the development of scenarios has much in common with describing the development process. If an organization does not have a well-defined process, scenario creation might be conducted in a kind of workshop with all stakeholders. Such a workshop facilitates the creation of a common understanding within the team and at the same time increases the acceptance of the scenarios within the team.

4.2. Example of the Scenario Development Process

We consider it important to provide an example to demonstrate the process of deriving a scenario. For this purpose, we use an object-oriented development process (i.e. Fusion). Our main goal is to demonstrate the feasibility of the development approach in the context of a specific inspection situation. Although we describe in detail how to develop the scenario for this situation, the scenario itself is more generic in the sense that it can be used for other inspections of the same type of information and documentation. The numbers used to explain the example reflect the numbers used in the process description of the previous section.

1. The first task in setting up a perspective-based reading scenario is to define precisely the information and the documentation that will be the subject of the inspection. In this example, it is the description of a system operation (i.e. the "validation_result" operation of a check out point of sales subsystem). The documentation of the operation includes an operation schemata, class diagrams, collaboration diagrams, and operation pseudocode. The following Figures 6 to 9 present examples of the diagram types for which scenarios can be developed. It is important to mention that the scenarios are specific to the diagram type and not to the information described in the diagrams.

2. The next step is to define the stakeholders that have an interest in the quality of the information in the documents, and thus represent a perspective from which to inspect it. Any person, or role, which is in some way affected by the information's quality, however remote, can serve as the basis of an inspection perspective. In this example we will consider the typical stakeholders used in perspective-based inspection, which are defined in terms of the roles in the development process.

Operation: validation_result
Description: Completes a payment by credit card

Reads: **supplied** valid : Boolean, **supplied** credit : integer
Changes:
Sends: display:{invalid, insufficient_credit}, till {open}
database {completed_sale}
Assumes:
Result:
if valid = true then
  if credit > bill.total then
    the message completed_sale(bill) has been sent to the database
    the message open has been sent to till
  else
    the message insufficient_credit has been sent to display
else
  the message invalid has been sent to display

Fig.6. Operation Schemata

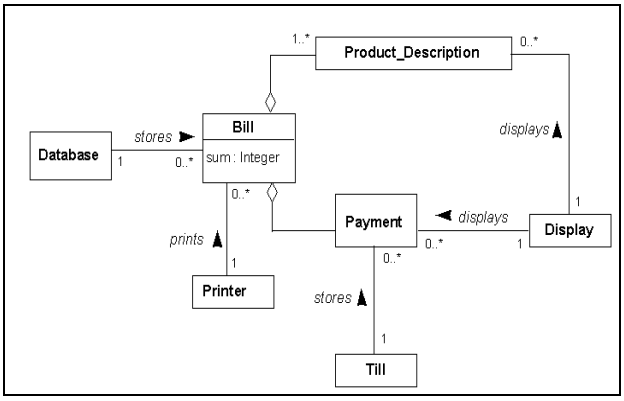

Fig.7. Operation Schemata

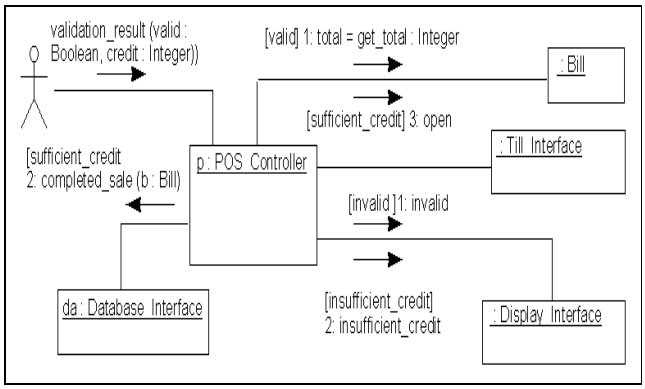

Fig.8. Collaboration Diagram

```
operation POS_System: validation_result (valid : Boolean, credit : Integer)
if valid = true then
    get the total from the Bill
    if credit > bill.total then
        tell the database interface to send the bill to the database
        tell the till interface to open the till
    else if credit = bill.total then
        tell the interface display to display insufficient_credit message
else
    tell the interface display to display the card invalid message
```

Fig.9. Operation Pseudocode

3. Hence, the stakeholders we could consider are requirements engineer, designer, tester, and maintainer.

Although the list of stakeholders is not exhaustive, it serves to illustrate the approach.

TABLE I
RELATIONSHIP BETWEEN DOCUMENT TYPES AND ROLES

	Require-ments Engineer	Designer	Tester	Maintainer
Operation Schema	√	√	√	
Class Diagram	√			
Collaboration Diagram		√		√
Operation Pseudocode		√	√	√

Table 1 demonstrates that some perspectives only check the quality among analysis documents while others check the mapping between analysis and design documents. This does not represent a clean separation between horizontal reading (i.e., checking the quality of models on one level of development) and vertical reading (i.e., checking the quality of models between levels of development). It rather demonstrates the coverage that can be achieved by selecting relevant perspectives.

3. In this example, we select the designer to illustrate the next steps.
4. We set up the introductory part of the scenario as follows:

   "Assume you are inspecting a system operation from the perspective of a a designer. The main task of a designer is to describe how the operation meets its responsibilities in terms of interactions between objects. High quality is determined by correctness of the design with respect to the specification, and the satisfaction of performance goals."
5. The next step is to identify the instruction part:

   "Locate the collaboration diagram, the pseudocode description and the schema for the operation. For each possible outcome of the postcondition, ensure that the appropriate messages are dispatched between the appropriate objects to achieve the desired goal. Mark the outcome as well as the message with a colored pen. Check that the outcomes and the messages described in the pseudocode and the collaboration diagram are consistent."
6. The question part of the scenario include the following ones:

   1. For every message defined in the operation schema, is there a corresponding message sent in the collaboration diagram?

   2. For every attribute that is changed in the operation schema, is an appropriate message sent to the corresponding object in the collaboration diagram?

   3. Are there any discrepancies between the algorithms defined in the collaboration diagram and the pseudocode description?

   4. Are the initial conditions for starting up a function clear and correct?

   5. Are the effects of a function specified under all possible circumstances?
7. For this scenario no additional documents need to be created.

Figure 10 present the complete scenario for reading the documentation of a system operation from the perspective of a designer.

Designer Scenario for the System Operation

Introduction:

Assume you are inspecting a system operation from the perspective of a designer. The main task of a designer is to describe how the operation meets its responsibilities in terms of interactions between objects. High quality is determined by correctness of the design with respect to the specification, and the satisfaction of performance goals.

Instruction:

Locate the collaboration diagram, the pseudocode description and the schema for the operation. For each possible outcome of the postcondition, ensure that the appropriate messages are dispatched between the appropriate objects to achieve the desired goal. Mark the outcome as well as the message with a colored pen. Check that the outcomes and the messages described in the pseudocode and the collaboration diagram are consistent.

Questions:

1. For every message defined in the operation schema, is there a corresponding message sent in the collaboration diagram?
2. For every attribute that is changed in the operation schema, is an appropriate message sent to the corresponding object in the collaboration diagram?
3. Are there any discrepancies between the algorithms defined in the collaboration diagram and the pseudocode description?
4. Are the initial conditions for starting up a function clear and correct?
5. Are the effects of a function specified under all possible circumstances?

Fig.10. Scenario for Reading the Documentation of a System Operation from the Designer's Perspective

8. For this scenario no additional forms and templates need to be developed for the defect scrutiny. The inspectors only need a standard form for documenting the defects detected. Examples of this kind of form can be found in [11].

4.3. Scenario Improvement

After the development of the reading scenarios the experiences of using them in the inspection process are a major source of information for their improvement. In addition quantitative information about the inspection process also indicates improvement opportunities for the defect detection activity. The improvements may be characterized and described according to the following dimensions:

- Number of Perspectives
  To determine and optimize the number of perspectives, the scenario developer needs to analyze the collected inspection data. In particular he or she needs to analyze the defect slippage, i.e., those defects that were not detected throughout an inspection. This situation can occur if important quality attributes for a specific group of stakeholders were omitted. The solution to this issue is an additional stakeholder perspective in an inspection. A rigorous defect inspection data analysis may also reveal that the quality checks of a particular perspective are already included in other ones. In this case, this particular perspective does not reveal a benefit to the inspection team results and , thus, can be omitted.
- Number of Scenarios
  In some cases the scope of the required activities for a single perspective may be too effort consuming for a single scenario. A larger than usual preparation effort for inspectors applying this scenario may indicate the situation. As a consequence the inspector may be overwhelmed with too many information and quality checks. Defects may therefore be not detected although covered by a particular scenario. A solution to this issue is to set up two scenarios that involve different activities for a particular stakeholder perspective. Each of these scenarios can be given to different inspectors to avoid overloading.
- Level of Specificity
  It may be necessary to change the level of specificity of the scenarios and provide a more or less detailed description of the activities. The necessity can be determined by interviewing the inspectors that work with the various scenarios. The feedback throughout the interviews can also be used to guide the scenario improvement.
- Level of Abstraction
  The chunk of information that an inspector needs to check may be too large. An indication for this is the amount of preparation effort each inspector spends. In this case it may be better to create a second scenario to split the work among inspectors.
- Experience of Inspectors
  Throughout the development project, the inspectors become increasingly familiar with the different scenarios. This may justify a change in the description of activities. Interviews with the inspectors help determine this situation and drive scenario improvement.
- New Problems in the Environment
  New issues, such as new defect classes, may pop up in the specific environment or new findings in the environment can make it necessary to change the scenario description (i.e., either the activities, the questions, or both). This depends on the specific situation and is related to the topic of software process improvement.
- Coverage
  A measurement program for inspections may reveal that some defects slip through the inspection process. Defect causal analysis can be used to refine the scenarios in a way that help inspectors detect these defects. This helps avoid future defect slippage.

All these dimensions with their suggestions help improve the PBR scenarios. Scenario improvement directly impacts inspection defect detection effectiveness and, thus, the cost of defects in a particular project or organization.

5. CONCLUSION

Software inspections belong to the most cost-effective methods for quality improvement and cost reduction. Despite their many benefits, they are not used on a broad basis in the software industry. Part of the problem is the human nature of inspections as well as their costs. Scenario-based reading techniques have been suggested to alleviate this problems. Several scenario-based reading techniques, such as the perspective-based reading technique have already been described and their benefits have been empirically validated.

Given the body of evidence demonstrating the cost-effectiveness of scenario-based reading in general and perspective-based reading in particular, one might have expected more comprehensive support on how to adapt this technique to project characteristics. However, this is rarely the case. This paper has filled this gap and presented a detailed process for tailoring perspective-based inspections to project specificities.

The process allows practitioners to develop an initial set of perspective-based reading scenarios and upgrade their existing inspection implementation with a more rigorous defect detection technique (i.e., to implement the perspective-based inspection approach). In this way, practitioners can leverage and amplify the cost-effectiveness of their inspections. This effect has already been demonstrated in a number of empirical studies. More information can be found at http:\\www.iese.fhg.de\Inspections .

REFERENCES

[1] Ackerman, A. F., Buchwald, L. S., and Lewsky, F. H., 1989. Software Inspections: An Effective Verification Process. IEEE Software, 6(3): 31-36.

[2] Adler, M.J., van Doren C., How to Read a Book, Simon and Schuster, New York 1972.

[3] Atkinson, C., Bayer, J., Bunse C., Kamsties, K., Laitenberger, O., Laqua R., Muthig, D., Paech, B., Wüst, J., Zettel, J., Product Line Engineering with the UML, Addison-Wesley Publishing Company, To appear in 2001.

[4] Basili, V., Green, S., Laitenberger, O., Lanubile, F., Shull, F., Sorumgard, S., and Zelkowitz, M., 1996. The Empirical Investigation of Perspective-based Reading. Journal of Empirical Software Engineering, 2(1):133-164.

[5] Basili, V. R., 1997. Evolving and Packaging Reading Technologies. Journal of Systems and Software, 38(1).

[6] Briand, L., El-Emam, K., Fussbroich, T., and Laitenberger, O., 1998. Using Simulation to Build Inspection Efficiency Benchmarks for Development Projects. Proceedings of the 20th International Conference on Software Engineering, pages 340-349.

[7] Briand, L. C., Freimut, B.G., Klein, B., Laitenberger, O., and Ruhe, G., 1998, Quality Assurance Technologies for the EURO Conversion - Industrial Experience at Allianz Life Assurance, in 2nd International Software Quality Week Europe, Brussels, Belgium.

[8] Fagan, M. E., 1976. Design and Code Inspections to Reduce Errors in Program Development. IBM Systems Journal, 15(3):182-211.

[9] Fowler, P. J., 1986. In-process Inspections of Workproducts at AT&T. AT&T Technical Journal, 65(2):102-112.

[10] Gilb, T. and Graham, D., 1993. Software Inspection. Addison-Wesley Publishing Company.

[11] Graden, M. E., Horsley, P. S., and Pingel, T. C., 1986. The Effects of Software Inspections on a major Telecommunications-project. AT&T Technical Journal, 65(3):32-40..

[12] Johnson, P.M., Tjahjono, D., 1998, Does Every Inspection Really Need a Meeting, Journal of Empirical Software Engineering, vol. 3, no. 1, pp. 9-35.

[13] Laitenberger, O., Cost-Effective Detection of Software Defects through Perspective-based Inspections, PhD-Thesis, University of Kaiserslautern, ISBN 3-8167-5583-6, 2000.

[14] Laitenberger, O. and DeBaud, J.-M., A Survey of Work in Software Inspections, Journal of Systems and Software, vol. 50, no. 1., 2000.

[15] Laitenberger, O. and DeBaud, J.-M., 1997. Perspective-based Reading of Code Documents at Robert Bosch GmbH. Information and Software Technology, 39:781-791.

[16] Laitenberger, O., El Emam, K., and Harbich, T., 2001, An Internally Replicated Quasi-Experimental Comparison of Checklist and Perspective-based Reading of Code Documents. IEEE Transactions on Software Engineering, 2001.

[17] Laitenberger, O., Atkinson, C., 1999, Generalizing Perspective-based Inspection to handle Object-Oriented Development Artifacts, in Proceedings of the International Conference of Software Engineering.

[18] Laitenberger, O., Kohler, K. Atkinson, C., Architecture-centric Inspection for the Unified Development Process (UP), Proceedings of the European Conference on Software Testing, Analysis and Review, 2000

[19] Miller, G.A., 1956, The magical number seven, plus or minus two: some limits on the capacity for processing information. The Psychological Review, 63(2):81-97.

[20] Parnas, D. L., 1987. Active Design Reviews: Principles and Practice. Journal of Systems and Software, 7:259-265.

[21] Parnas, D. L. and Weiss, D., 1985. Active Design Reviews: Principles and Practices. Proceedings of the 8th International Conference on Software Engineering, pages 132-136. Also Available as NRL Report 8927, 18 November 1985.

[22] Porter, A. A. and Johnson, P. M., 1997. Assessing Software Review Meetings: Results of a Comparative Analysis of Two Experimental Studies. IEEE Transactions on Software Engineering, 23(3):129-144.

[23] Porter, A.A., Votta, L., 1998. Comparing Detection Methods for Software Requirements Inspection: A Replication using Professional Subjects, , Journal of Empirical Software Engineering, vol. 3, no. 4, pp. 355-378.

[24] Porter, A. A., Votta, L. G., and Basili, V. R., 1995. Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment. IEEE Transactions on Software Engineering, 21(6):563-575.

[25] Travassos, G., Shull, F., Fredericks, M., and Basili, V.R., 1999. Detecting defects in object oriented designs: Using reading techniques to increase software quality. In the Conference on Object-oriented Programming Systems, Languages & Applications (OOPSLA).

[26] Votta, L. G., 1993. Does Every Inspection Need a Meeting? ACM Software Eng. Notes, 18(5):107-114.