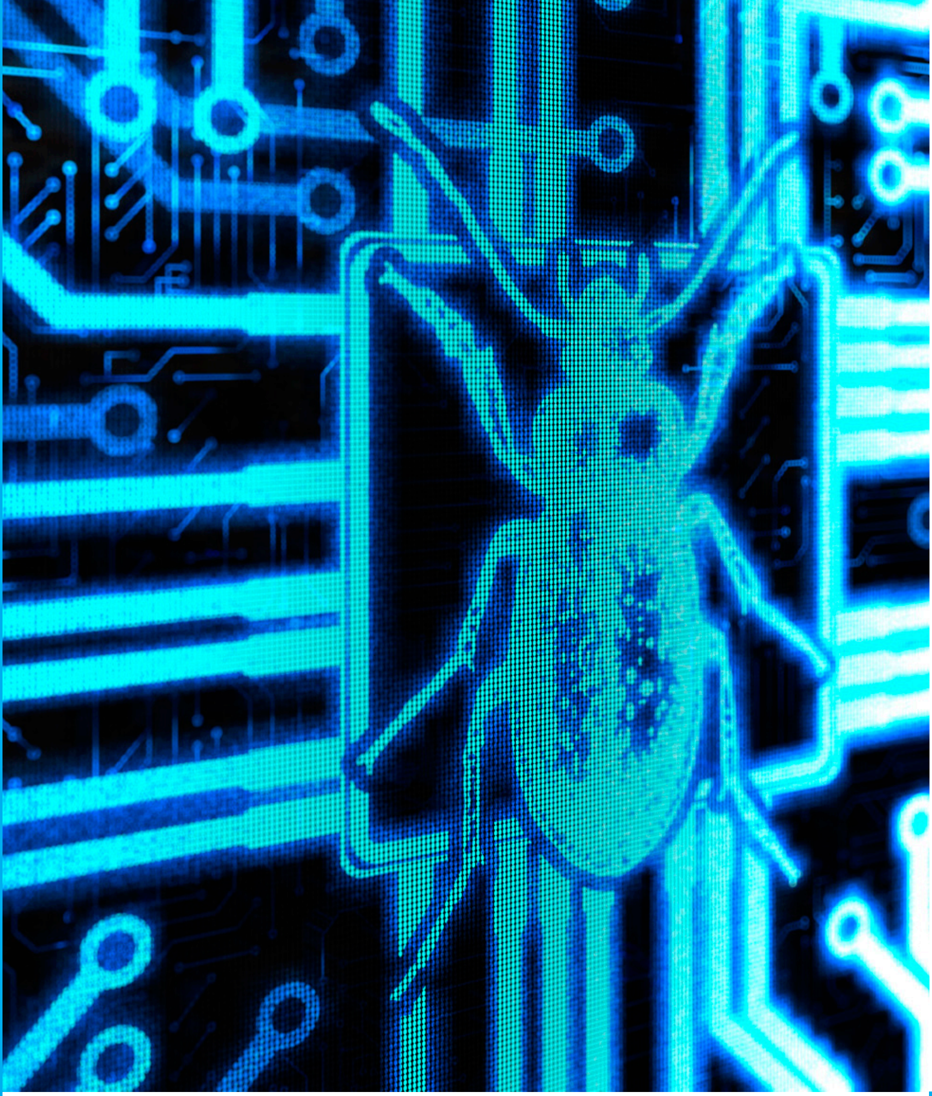


TECHNOLOGY • RESEARCH HIGHLIGHTS • VISIONS • SCIENCE

9

Application of the perspective-based reading technique in the nuclear I&C context

CORSICA work report 2011

Jussi Lahtinen

VTT TECHNOLOGY 9

Application of the perspective-based reading technique in the nuclear I&C context

CORSICA work report 2011

Jussi Lahtinen

ISBN 978-951-38-7621-0 (URL: http://www.vtt.fi/publications/index.jsp)
ISSN 2242-122X (URL: http://www.vtt.fi/publications/index.jsp)

JULKAISIJA – UTGIVARE – PUBLISHER

VTT
PL 1000 (Vuorimiehentie 5, Espoo)
02044 VTT
Puh. 020 722 111, faksi 020 722 4374

VTT
PB 1000 (Bergsmansvägen 5, Esbo)
FI-2044 VTT
Tfn +358 20 722 111, telefax +358 20 722 4374

VTT Technical Research Centre of Finland
P.O. Box 1000 (Vuorimiehentie 5, Espoo)
FI-02044 VTT, Finland
Tel. +358 20 722 111, fax + 358 20 722 4374

Application of the perspective-based reading technique in the nuclear I&C context
CORSICA work report 2011

Jussi Lahtinen. Espoo 2012. VTT Technology 9. 45 p. + app. 7 p.

Abstract

Inspections and reviews are one of the most effective ways of detecting errors in software development. The methods are also cost-effective because defects can be spotted early in the development, and thus the cost of repairing the defects is lower.

Reading techniques are the procedures that are used in the inspection or review of a software artefact. The most common procedures are simple ad-hoc reading and a checklist-based reading technique. However, more advanced and detailed procedures have been created for various purposes.

This report reviews the state-of-the-art software reading techniques used in inspections and reviews, and briefly reviews some of the empirical research in this context. The majority of the empirical research results indicate that, for example, perspective-based reading is more cost-effective and can detect more defects than more basic reading techniques.

This report also describes how perspective-based reading can be applied to the inspection of nuclear-domain requirement specifications. For this purpose, seven perspective-based reading scenarios have been created.

Keywords software inspection, review, reading technique, perspective-based reading, nuclear, requirements specification

Contents

1. Introduction

Inspection is a well-defined process used for defect detection in software projects. Typically, inspected software artefacts include requirements, design documentation, test plans, and code. Similar techniques used in software projects include walkthroughs and reviews. For clarity, we refer to the definition in the IEEE Standard 1028-2008 [IEEE, 2008], which provides the following descriptions:

- An **inspection** is 'a visual examination of a software product to detect and identify software anomalies, including errors and deviations from standards and specifications.'
- A **walkthrough** is 'a static analysis technique in which a designer or programmer leads members of the development team and other interested parties through a software product, and the participants ask questions and make comments about possible anomalies, violation of development standards, and other problems.'
- A **review** is 'a process or meeting during which a software product, set of software products, or a software process is presented to project personnel, managers, users, customers, user representatives, auditors or other interested parties for examination, comment or approval.'

The word inspection is sometimes used to refer to a Fagan inspection, the first formally defined inspection technique, which has been used as a model for many subsequent inspection techniques. Fagan's technique [Fagan, 1976] was the first formally defined inspection technique. The technique is based on a team of reviewers following a step-by-step procedure (Figure 1).

The inspection team members play roles according their skills and knowledge. The team member roles defined by Fagan are: moderator, author, reader, and reviewer. The moderator manages the inspection team and coordinates the inspection process. The author is the programmer who is responsible for the work product under inspection. A reader is a person who paraphrases the work product during the meeting. A reviewer is a person who reviews the work product. A team member may play several roles.
Six steps are defined in the Fagan inspection procedure, and they are as follows:

- **Planning**. The material under inspection is determined, and the responsibilities of the inspection team are set.

- **Overview**. An overview of the product is presented to the inspectors. The scope of the work is determined.
- **Preparation**. The reviewers go through the material individually with the goal of understanding it thoroughly.
- **Inspection meeting**. The inspection team assembles. The reviewers state their findings, which are recorded by the moderator.
- **Rework**. The author reassesses and modifies the work product. The rework is verified by holding another inspection.
- **Follow-up**. The moderator ensures that defects have been repaired.

Novel inspections are typically modified versions of the Fagan inspection. A particular software inspection can be characterised using a taxonomy of inspection approaches that has four dimensions: the technical, economic, organisational, and tool dimensions [Laitenberger, 2002]. Most research in the software inspection area focuses on which of the aspects in these dimensions has a positive influence on the inspection results.

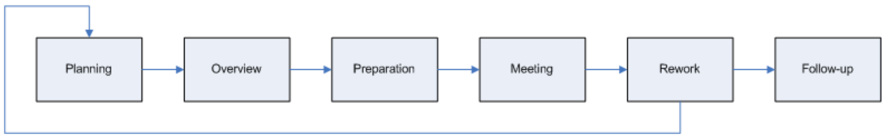


Figure 1. The Fagan inspection process.

The technical dimension describes how the inspection is tailored for a particular situation. The technical dimension includes the inspection process (e.g. in Figure 1), the inspected artefact, the roles of the participants, team size, and the applied reading technique.

The economic dimension deals with evaluating whether the inspection is worth commencing. This includes the aspects of quality, effort, and duration of the inspection.

The organisational dimension includes organisational issues that affect the outcome of the inspection. These are the members of the inspection team, the project structure, culture, management, budget, quality, and productivity goals.

The tool dimension represents the tool support used for the inspection. For example, there are some software tools that support defect detection or document handling.

1.1 Inspection benefits

The effectiveness of inspections in detecting defects has been evaluated by many researchers. As an example:

- Fagan reported in his work that inspections detected 93% of all defects in a program by IBM. In two other projects, the effectiveness of inspections was over 50% [Fagan, 1986].
- The code inspections used at HP typically found 60% to 70% of the defects [Grady and van Slack, 1994].
- Inspections used in many occurrences indicated that the effectiveness of code inspections was typically in the range of 30% to 75% [Barnard and Price, 1994].

In addition to the effectiveness of defect detection of software inspections, the effort required to perform inspections is rather low when compared to other defect detection methods, such as testing. The cost of detecting and correcting a defect during inspections is much lower than detecting and correcting the defect during the testing phase of the product. The effort required per found defect has been widely studied. As an example:

- The ratio of fixing defects during inspection to fixing defects during formal tests varies from 1:10 to 1:34 according to [Kaner, 1998], 1:20 according to [Remus, 1984], and 1:13 according to [Kan, 1995].
- [Weller, 1993] reports that the time needed per defect in inspection is 1.43 hours (6 hours per defect in testing).
- [Kelly, 1992] found that, on average, 1.75 hours are required per defect in design inspections; 1.46 hours in code inspections; and 17 hours per defect in testing.

In conclusion, a majority of studies indicate that inspections are very effective in detecting defects, and that the cost of defect correction, when using inspections, is much lower than if the defect was found in a later developmental phase.

1.2 Improving inspections

The influence of modifications on various inspection aspects has been studied. Studies such as [Porter and Johnson, 1997] indicate that typical meeting-based review methods are neither more effective nor less effective than non-meeting-based review methods with respect to defect detection effectiveness. In fact, the non-meeting inspections found more defects, but there was no significant difference. In addition, the size of the inspection team and the coordination style of the inspection do not, apparently, increase the effectiveness of inspections [Porter and Votta, 1997].

Instead, explicit training in program understanding improves inspection effectiveness [Rifkin and Deimel, 1994]. It seems that individual preparation for inspections is the most important element contributing to the effectiveness of the inspection [Christenson, 1990; Laitenberger et al., 2002]. Defect detection is more an individual than a group activity, and the strategies that the individual inspectors use to understand and examine the artefact have great influence on the inspection results. Thus, advanced reading techniques that guide the individual preparation process can be useful in increasing defect detection effectiveness.

This work focuses primarily on one aspect of inspections: the reading techniques. State-of-the-art reading techniques are reviewed, and one reading technique called perspective-based reading is applied to the inspection of requirements specifications in the nuclear context.

2. Reading techniques

Based on experimental studies in [Porter and Johnson, 1997], it is concluded that typical meeting-based review methods are neither more effective nor less effective than non-meeting-based review methods with respect to defect detection effectiveness. In fact, the non-meeting inspections found more defects, but there was no significant difference.

The critical view regarding meetings is not consistent among researchers. Some researchers emphasise the importance of meetings in learning and sharing knowledge. In the study that found meetings ineffective [Porter and Johnson, 1997], the meetings still succeeded in eliminating more false positive defects.
In [Porter and Votta, 1997], it is also stated that the organisation of the inspection (inspection team size, inspection coordination) does not significantly increase the inspection effectiveness. Instead, as is noted in [Rifkin and Deimel, 1994], explicit training in program understanding may improve inspection effectiveness. Several studies [Christenson, 1990; Laitenberger et al., 2002] have reported results that support the claim that individual preparation for inspections is the most important element contributing to the effectiveness of the inspection. Defect detection is more an individual than a group activity, and the strategies that the individual inspectors use to understand and examine the artefact have a great influence on the inspection results. Thus, advanced reading techniques that guide the individual preparation process can be useful in increasing defect detection effectiveness.

A reading technique is a set of instructions given to the inspector in order to guide the inspection process. A reading technique can also be thought of as a defect detection strategy. The most popular reading techniques are ad-hoc reading and checklist-based reading.

2.1 Ad-hoc reading

Ad-hoc reading does not give any guidance for inspectors. The inspector simply attempts to find as many defects as possible by examining the document or artefact using the skills and knowledge he/she has. It is evident that the ad-hoc technique is very dependent on the individuals performing the inspection. Ad-hoc reading is the most common reading technique used in inspections.

2.2 Checklist-based reading

In checklist-based reading (CBR), the inspector is given a list of questions that are to be answered during the inspection. The questions are written to draw the attention of the inspector to some aspects of the inspected artefact that are often found defective. Checklists give support to the inspector, and the result of the inspection is not as dependent on the skills of the individual inspectors as in ad-hoc reading. One major disadvantage of using CBR is that only defects of a particular type are detected (defects detectable by answering the questions). Thus, hard-to-find defects that can be found through a deep understanding of the artefact are often missed. The CBR technique can only find errors that have been previously encountered or thought of as the list of inspection questions is written.

Another disadvantage is that the checklist is typically quite generic, and some of the questions might not be suitable for the artefact. CBR is typically also missing instructions on how the questions should be answered (i.e. should the inspector read through the document once and then answer the questions, or examine the document while considering a single question at a time). Finally, in CBR techniques, the reviewers are overloaded with excessive information as the checklists tend to be extensive, and the reviewer must go through all of the reviewed documentation.

2.3 Reading by stepwise abstraction

Cleanroom [Selby et al., 1987] is a software engineering process developed at IBM that focuses on defect prevention to produce highly reliable software. Only a small part of the Cleanroom process incorporates reading techniques. The Cleanroom process relies on formal methods, incremental implementation, development without program execution and statistical testing. In Cleanroom, the testing process is completely separated from the development process. Thus, the developers rely on other techniques to assert the correctness of their implementation. One of these integrated techniques, verification-based inspection, is described in [Dyer, 1992]. The inspection technique described in this paper forms an informal proof of the correctness of a code artefact carried out in conversation or in writing. The reading technique is known as ‘reading by stepwise abstraction’.

In reading by stepwise abstraction, the inspector decomposes each function into a set of component functions, and then derives the function of the entire program from these component functions. The inspector starts from the individual code statements, and abstracts the functions that these statements compute. Then the inspector selects a higher-level structure of the code and abstracts it using the previous reasoning. The procedure is repeated until the final function has been abstracted and articulated, and can be compared against the specification of the program. Reading by stepwise abstraction is limited to the inspection of code artefacts.

2.4 Active design reviews

Active design reviews [Parnas and Weiss, 1985] is an inspection technique for design artefacts that is based on the following ideas:

1. Reviewers should focus on aspects that suit their expertise.
2. Desired reviewer characteristics should be specified and the reviewers selected based on these specifications.
3. In addition to defect detection, the reviewers should be encouraged to make positive assertions about the design (i.e. documenting positive observations as well).
4. The designer asks questions from the reviewer.
5. Designers and reviewers conduct small meetings of 2–5 people.

In active design reviews, the reviewers are chosen so that their expertise covers the design completely. One reviewer can then work more effectively by focusing on subjects relevant to their expertise and ignoring the irrelevant parts. It is also important that the reviewers think hard what they are reading. The reviewers are forced to take an active role by asking questions that can only be answered by careful study of the document. The reviewers may also be asked to write a small program that implements some part of the reviewed design.

The active design review process has three steps. The first step is an overview of the design and schedule. In the next step, the reviewers go through the design and answer the questions formulated by the designer. In the third step, the designers read the answers by the reviewers and discuss together until comprehension of all issues is reached.

2.5 Scenario-based reading techniques

A paper by Porter and Votta [Porter and Votta, 1994] launched an active period of research on reading techniques. In the paper, a reading technique titled ‘scenario-based reading’ is introduced. Later on, the term ‘scenario-based reading’ was defined as a general-level term, and the technique described in the paper is usually referred to as ‘defect-based reading’. Variations on the original technique can be referred to as scenario-based techniques. These techniques have much in common with the principles of the active design review technique. Scenario-based reading techniques are based on scenarios that give customised guidance for inspectors. The guidance is more specific than the instructions given in checklist-based reading. It may be a set of questions, an assignment, or explicit instructions on how to perform the review. A scenario also draws the inspector’s attention to only a particular type of defect. Each inspector typically uses a different scenario, and it is assumed that the overall team will be more effective, detecting fewer overlapping defects.

2.5.1 Defect-based reading

Defect-based reading [Porter and Votta, 1994] and [Porter et al., 1995] was the first scenario-based reading technique that was used to review software requirements specification documents. The technique gives the reviewers specific orthogonal detection responsibilities (i.e. scenarios) that are derived from particular classes of faults. In [Porter et al., 1995] the scenarios are derived from checklist questions used to review documents. In the paper, the three different scenarios were based on three fault classes: 1) data type inconsistencies, 2) incorrect functions, and 3) missing or ambiguous functions. A scenario consists of a set of questions that the reviewer must answer while examining the document. Simplified examples of the scenario questions are:

- Identify all data objects. Are all data objects mentioned in the overview listed in the external interface section? (**Data type consistency scenario**)
- Identify all specified system events. Is the specification of these events consistent with their intended interpretation? (**Incorrect functionality scenario**)
- Identify the required precision, response time, and so on for each functional requirement. Are all required precisions indicated? (**Ambiguities or missing functionality scenario**)

Each reviewer applies a single scenario (looks for one fault class only) and the reviewers together achieve sufficient coverage of the document. [Porter et al. 1995] found in their studies that defect-based reading techniques detected 35% more defects than reviewers applying ad-hoc or checklist-based reading techniques. Defect-based reading helped concentrate on certain defects but was not less effective at detecting other defects. The experiment results also indicated that checklist-based reading was no more effective than ad-hoc reading.

In a controlled experiment, defect-based reading was compared to CBR and ad-hoc reading for inspection of software requirements specifications. The defect-based reading technique had a higher defect detection rate (improvement of 35%) than the other methods. The experiment was replicated using a partial factorial randomised experimental design. The replication also indicated that the defect-based technique had a higher fault detection rate than the other methods [Porter, 1994 and Votta; Porter et al., 1995]. These results were obtained using graduate students as test subjects. Later, the results were also confirmed using professional software developers [Porter and Votta, 1998].

2.5.2 Scenario-based reading based on function point analysis

[Cheng and Jeffrey 1996] also studied the inspection of software requirements specifications and hypothesised that other strategies for scenario partitioning might perform as well as partitioning based on fault classes. They assumed that the effectiveness of the scenario-based approach was the consequence of a reduction in work load. Their decomposition of scenarios was based on a function point analysis (FPA). In FPA, software is seen as a collection of internal logical files, external interface files, external inputs, external outputs, and external inquiries. Using this classification, the software can be divided into orthogonal areas: files, inputs, outputs, and inquiries. These areas are constructed into function point scenarios that investigate different aspects of the software requirements specification:

1. Overview scenario
2. File scenario
3. Input scenario
4. Output scenario
5. Inquiry scenario.

Again, each scenario consists of questions that focus on specific function-point items.

The created scenario-based reading technique was compared to an ad-hoc approach in which the reviewers were required to develop their own inspection strategy prior to the inspection. The results showed that the self-set strategy was more effective in detecting defects than the approach based on function-point scenarios, but the difference was not statistically significant. The result still supports the claim that experienced people might be able to create inspection strategies that are better than a provided set of scenarios.

In [Cheng and Jeffrey, 1996], scenario-based reading based on function point analysis was compared to self-set inspection strategies to inspect software requirements specifications. No significant difference between the two groups was found in their inspection performance.

2.5.3 Perspective-based reading

In the perspective-based reading (PBR) technique software artefacts are examined from the perspectives of the artefacts' stakeholders in order to identify defects. The technique is covered in detail in Chapter 3.

2.5.4 Perspective-based usability inspection

The perspective-based method has also been applied to the inspection of software usability [Zhang et al., 1998]. Traditional usability inspection techniques such as heuristic evaluation, cognitive walkthrough, and pluralistic walkthrough have a rather low problem detection rate. In perspective-based inspection, different inspection sessions focus on a subset of usability issues covered by one of several usability perspectives. The perspectives provide a point of view, and a list of questions that represent usability issues and a procedure for the inspection.

The perspectives defined in the paper [Zhang et al., 1998] were: novice use, expert use, and error handling. The novice use perspective handled issues concerning users with very little experience of using the system. The second perspective, expert use, dealt with issues related to the efficiency, flexibility and consistency of the system. For the error handling perspective, the inspectors tried to come up with possible errors in the system.

The perspective-based usability inspection method was evaluated in an experiment. The method was compared to a control group using heuristic evaluation to find usability problems in a web-based data collection form interface. The improvement of the perspective-based technique over the heuristic evaluation technique was 30% on average. The focused attention helped inspectors to find more problems in certain categories without the total number of problems found suffering [Zhang et al., 1998].

2.6 Scope-based reading

Many reading techniques are used only for defect detection. However, reading techniques can also be applied to improve learning. For example, [Shull, 1998] introduces the scope-based reading (SBR) technique. Scope-based reading is a reading technique that is used to help software developers construct designs from object-oriented frameworks. The technique is used as a tool to understand what a system does by abstracting the important information in the system and identifying relevant reusable portions.

The system that the techniques were applied to in the paper [Shull, 1998] consisted of a class hierarchy and a model that describes how objects derived from the class hierarchy are meant to interact with each other. The framework can then be used to solve a particular problem. The purpose of the SBR reading techniques was to enhance software developers' understanding as they studied a framework that they had not used before.

The user of an SBR technique is given instructions that are based on two procedures: an abstraction procedure and a use procedure. The abstraction procedure discusses how the functionality of the framework is understood. The use procedure describes how reusable functionality can be identified in the framework.

Two separate reading techniques were introduced by Shull: the hierarchy-based technique and the example-based technique.

In the hierarchy-based technique, the framework is described through the class hierarchy and an object model that describes how dynamic behaviour is implemented. The abstraction procedure is based on this description. The inspectors concentrate on abstract classes in the class hierarchy and then iterate through levels of concrete classes to find the most specific class for a particular task. In the use procedure, the relevant classes are found, and it is deduced whether a single class can be used to implement a particular behaviour or if several classes should be combined. In the paper [Shull, 1998], the hierarchy-based technique was assumed to give the user a broader knowledge of the framework than the example-based technique.

In the example-based technique, the framework is looked at through a set of example applications that demonstrate the functionality provided by the framework. The created abstraction procedure guides the user through the levels of detail within the set of examples at varying levels of complexity. The user selects examples from the example set and identifies the objects and methods in these examples that are responsible for the implementation in the example. The use procedure of the example-based technique consists of identifying the examples that are likely to contain reusable functionality and identifying the potentially relevant classes and methods in those examples. The example-based technique was assumed to concentrate more on the specific functionality of the framework that were most relevant to the particular task at hand. The example-based technique was found to be very useful, especially in introducing novice users to the framework.

2.7 Usage-based reading

Usage-based reading (UBR) [Thelin et al., 2001] is a technique for design inspections. The idea behind the technique is that not all faults are equally important. In UBR, the goal is not to find as many faults as possible, but to find the most critical faults. Thus, UBR focuses the inspector's effort on detecting faults that most negatively affect system quality if not fulfilled.

The inspection technique of UBR is as follows. The inspector has a set of use cases that have been developed during requirements specification. Then the inspector traces and manually executes the use cases on the inspected design, using the requirements specification as a reference. During the inspection, the inspector ensures that the design fulfils the goal of the use case, and that all relevant functionality is provided.

In order to focus the user's perception on the most critical design aspects, the use cases are prioritised. The prioritisation can be done by a group of users using pair-wise comparisons according to the analytic hierarchy process [Saaty and Vargas, 2001]. Using the priorities of the use cases, the inspection can then follow either a ranked-based or a time-controlled procedure. In ranked-based reading, the use cases are gone through starting from the use case with the highest priority. In time-controlled reading, the reviewer divides the available time between the use cases so that higher ranked use cases receive more time.

Reviewers using usage-based reading are significantly more efficient and effective in detecting the most critical faults than reviewers using checklist-based reading [Thelin et al., 2003].

2.8 Traceability-based reading

A reading technique for inspecting high-level UML design diagrams is presented in [Travassos et al., 1999a; Travassos et al., 1999b]. The technique, called traceability-based reading (TBR) focuses only on inspecting high-level object-oriented designs represented in UML. The purpose of the presented reading technique is to verify that design diagrams are consistent among themselves and that they capture the requirements adequately.

Defect taxonomy is used and applied to the object-oriented designs to detect defects from the UML diagrams. The taxonomy defines five defect types: omission, incorrect fact, inconsistency, ambiguity, and extraneous information.

The major difference in TBR when compared to other reading techniques is that, for checking the correctness of a design, two separate reading tactics (horizontal and vertical reading) are performed. Horizontal reading verifies that the design diagrams are correct and consistent among themselves. Vertical reading ensures that the design diagrams are consistent with respect to the system's functional requirements and use cases. The two techniques are both defined within the family of traceability-based reading techniques.

Horizontal reading refers to reading techniques that are used to read and compare documents built in the same software life-cycle phase. A specific reading technique is defined for each pair of diagrams that can be usefully compared to each other. In the paper [Travassos et al., 1999b], the presented horizontal techniques are basically syntactical comparisons of:

- Class diagrams with respect to class descriptions
- Class diagrams with respect to state machine diagrams
- Sequence diagrams with respect to state machine diagrams
- Sequence diagrams with respect to class diagrams.

As an example, sequence diagrams illustrate messages between objects, while state diagrams show how the system responds to events that can be messages, or services of functions. The idea of horizontal reading is to help reduce the semantic gap between different documents by exploring these differences.

While horizontal reading identifies whether all of the design diagrams describe the same system, vertical reading verifies whether the design diagrams represent the right system, which is specified in the requirements and use cases. [Travassos et al., 1999b] describe the following vertical reading techniques:

- Comparing class descriptions with respect to textual requirements
- Comparing sequence diagrams with respect to use cases
- Comparing state machine diagrams with respect to textual requirements and use cases.

The guidelines given to the inspector given in every (horizontal or vertical) reading technique consist of instructions on how to use a certain marking system to highlight some concepts in the diagrams using pen and paper. After this, defects are identified by detecting discrepancies among the markings.

The authors of [Travassos et al., 1999b] performed a feasibility study on the TBR techniques and found that both horizontal and vertical reading techniques were necessary in order to cover all defects in the design diagrams. Subjects using vertical reading reported, on average, slightly more defects of omission and incorrect facts, while subjects using horizontal reading tended to report more defects of ambiguity and inconsistency.

2.9 Abstraction-driven technique

The performance of inspections is suggested to suffer from the delocalised nature of the software. This means that, in order to understand some parts of the code artefact, such as a line of code or a class, one has to have a previous knowledge of other methods, classes or libraries that are not part of the code under inspection. Because of this delocalisation, the inspection of object-oriented code documents can be troublesome.

As one solution to this problem, [Dunsmore et al., 2001] proposes that a specific reading technique could make the inspection of such documents more efficient. The resulting technique, titled the abstraction-driven technique, provides a strategy for reading code in a certain order, and a procedure for improving the inspector's understanding of the code:

- **Reading order guidelines**. The code of the entire system is first analysed, and the classes with the fewest interdependencies are inspected first. In a similar way, the methods within the classes are analysed, and methods with the fewest interdependencies are inspected first.
- **Procedure for improving understanding**. As the code is read, the inspector shall reverse engineer each method and class, and write an abstract specification for each method. The specification should be brief, in natural language, and complete. The authors of [Dunsmore et al., 2001] propose that the same abstraction technique that is used in the reading by stepwise abstraction technique (Section 2.3) can be used to come up with the specifications. While reading a method, all references to external classes shall be traced and understood.

An experimental investigation was also conducted in [Dunsmore et al., 2001], in which the systematic abstraction-driven technique was compared against ad-hoc reading techniques. As a result, the abstraction-driven technique was slightly more effective, but no significant difference was found. However, the application of the abstraction-driven technique appeared to find the delocalised defects better, and to help the weaker test subjects in detecting defects. Additionally, the abstraction-driven technique found more defects as the test subjects first ran through the code, reducing the need to re-read methods.

In an empirical investigation, the abstraction-driven reading technique was compared to ad-hoc techniques to detect defects in object-oriented code [Dunsmore et al., 2001]. The abstraction-driven technique was slightly more efficient, but the difference was not statistically significant.

2.10 Task-directed software inspection

Task-directed inspection (TDI) [Kelly et al., 2004] is a reading technique that was developed for the inspection of safety-related industrial legacy software in a particular environment. The technique includes a procedure for inspecting code modules for defects, and verifying their consistency with respect to documentation. Another part of the technique is a set of features related to the inspection process.

Task-directed inspection includes three inspection tasks that are to be performed on each code module. The tasks are broadly described below:

- **Task 1.** Create a data dictionary for the module that includes definitions, units, and a meaning for each variable. Verify that the variable is used in a corresponding manner.
- **Task 2.** Write a description of the code's logic and insert the description as a comment to the code.
- **Task 3.** Create a cross-reference between the code and the specifications in the documentation (manual) of the code. Create cross-reference tags and embed the tags to both the code and the documentation. Report mismatches.

Some aspects of the task-directed inspection technique resemble scenario-based reading. Some peculiarities of the TDI technique related to the inspection process are listed below:

- Individual work is emphasised, with no Fagan-style inspection meetings.
- Work is assigned to inspectors with knowledge of the particular domain. There is typically only one inspector per code module.
- The same person performs all of the inspection tasks on the same module.
- The inspection coordinator role is significant.
- Individual inspectors perform their work at their own pace whenever they have time, resulting in efficiency and low costs.

[Kelly et al., 2004] performed a case study using the TDI technique in an industrial setting. An inspection of 50 000 lines of code resulted in 950 findings, of which 6% were serious defects. The amount of documented code increased significantly and the users were satisfied with the outcome.

3. Perspective-based reading (PBR)

Perspective-based reading (PBR) [Basili et al., 1996] is a reading technique used in software inspections. The goal of perspective-based reading is to examine a software artefact description from the perspectives of the artefact's stakeholders in order to identify defects.

When compared to non-procedural reading techniques such as ad-hoc reading, the PBR technique can be characterised as:

- **Systematic.** The specific steps of the review process can be defined.
- **Focused.** Different reviewers focus on different aspects of the document.
- **Customisable.** PBR is customisable depending on the project and organisation. The PBR technique does not formalise a certain set of inspection procedures used for every possible software artefact, but instead instructs how the perspectives and procedures can be created based on the software artefact at hand.
- **Allowing controlled improvement.** Based on experience from previous reviews, the scenarios used can be improved by modifying the questions that are part of the scenarios.
- **Allowing training.** The specific review procedure allows the training of reviewers because the technique does not rely on the reviewer's experience with recognizing defects [Shull et al., 2000].

Perspective-based reading has been applied to various software documents. At least requirements documents [Basili et al., 1996], design models [Laitenberger and Atkinson, 1999], and code documents [Laitenberger and DeBaud, 1997] have been inspected using PBR. PBR techniques are expected to reduce human influence on the inspection results, and increase the cost-effectiveness of the inspections. The empirical research on the PBR technique is discussed in Section 3.7. Most research papers indicate that PBR is significantly more efficient and cost-effective than traditional reading techniques (ad-hoc and checklist-based reading).

3.1 Perspectives

One main idea of the perspective-based reading technique is the same idea as in all scenario-based reading techniques: to inspect a document from different reviewer perspectives. In PBR, the perspectives are derived from the stakeholders of the document, that is, the most relevant people that actually use the inspected artefact during its life cycle. The reasoning behind this is that a document is probably of high quality when potential stakeholders that use the document cannot detect any defects in it [Shull et al., 2000].

Typical perspectives for an inspection of a requirements specification document could include a designer's perspective, a tester's perspective and a user's perspective. This is because the people with these perspectives (a designer, a tester, and a user) are probably the most relevant stakeholders that actually use the requirements specification at some point in the product's life cycle.

In practice, the higher quality achieved by reviewer perspectives manifests itself in higher defect coverage, and a more in-depth analysis of the faults. These consequences of reviewer perspectives have also been analysed empirically. The results of [Basili et al., 1996] and [Robbins and Carver, 2009] indicate that the use of reviewer perspectives provides a better coverage of found defects, and that the perspectives correspond to the kind of faults the reviewer finds (a reviewer using a certain perspective tends to find more defects related to that perspective). Because different perspectives view different aspects of the document as important, the review group together can achieve higher overall coverage of the defects in the document. The number of overlapping defects found by different reviewer perspectives should also decrease.

Furthermore, because each reader is responsible for only a narrow focused view of the document, any potential errors are analysed more rigorously. The in-depth aspect becomes more effective when the perspective of an individual reviewer is selected in such a way that it allows the reviewer to use their special knowledge and perform in a way that naturally suits them.

3.2 Reviewer work products

Another key characteristic of the PBR method is the active role of reviewers in the inspection. The idea is that the reviewer creates a high-level version of a work product that the user would normally create from their perspective. For example, a reviewer working from a tester's perspective could create a high-level test plan for the system or part of the system. A reviewer working from a designer's perspective could create a high-level design model. A user perspective work product could be a user manual or a set of use-cases.

By creating work products based on the reviewed document, the reviewer is forced to actually think from the given perspective. The intention is that, by producing work products themselves, the reviewers obtain a more profound understanding of the system, and thus are able to detect more defects that are difficult to find and not just superficial errors. These work products might also be used at later stages of the life cycle of the system so that the work does not go to waste.

In PBR, the high-level work products are used to analyse whether the reviewed document conforms to them. As the reviewer creates a work product, the reviewer answers a set of questions designed to point out various possible defects in the reviewed document [Shull et al., 2000].

3.3 Cognitive analysis of PBR

The cognitive processes used in the PBR technique were analysed in [Robbins and Carver, 2009]. They analysed PBR using the protocol analysis method that is used to analyse the cognitive processes used in problem-solving tasks. In protocol analysis, the reviewer thinks aloud through their work and this data is then collected and analysed. The analysis helps understanding of the cognitive processes and memory usage needed to perform the task.

The cognitive analysis led to several conclusions:

- The cognitive process (named defect trigger) of combining knowledge usually precedes defect detection.
- The found defects were classified a posteriori based on the perspective that is most likely to find the defect. Using a statistical test, it was then concluded that reviewers found significantly more defects of the type associated with their perspective.
- Previous experience related to the reviewer's perspective leads to a statistically significant increase in the use of that knowledge. Other domain knowledge (for example, knowledge of the system that the document concerns) does not correlate with the use of that knowledge.

3.4 Scenarios

A PBR scenario is a document of instructions, typically only a few pages long. For each perspective, one or more scenarios are written that consist of specific and repeatable actions that the reviewer has to perform, and a set of questions that the reviewer should answer. As described earlier, the actions are related to producing high-level work products to gain an understanding of the product from a particular perspective. Questions about the activity or the work product are then answered to identify potential defects.

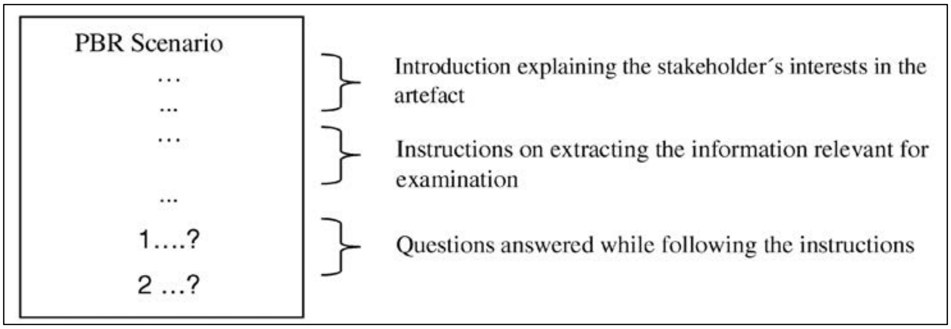

PBR Scenario
...
...
...
...
1....?
2 ...?
Introduction explaining the stakeholder´s interests in the artefact
Instructions on extracting the information relevant for examination
Questions answered while following the instructions

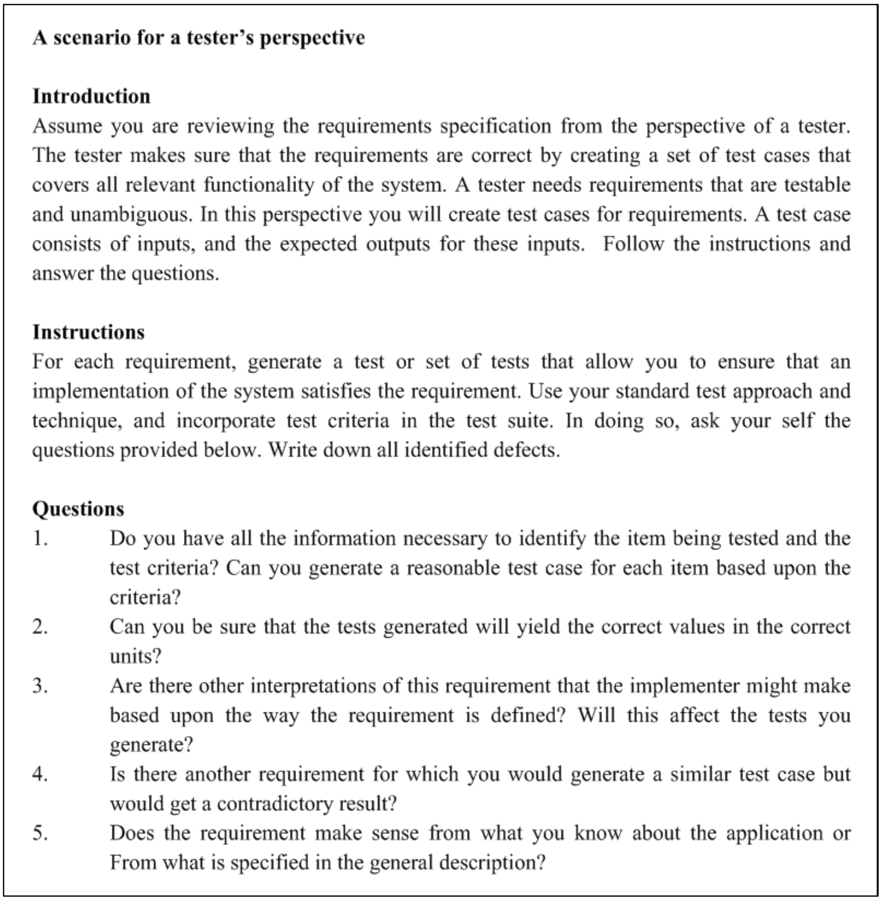
Figure 2. The PBR scenario structure (from [Laitenberger, 2000]).

The basic PBR scenario structure is illustrated in Figure 2. A scenario consists of three parts: an introduction, instructions, and questions:

- The introduction part explains the stakeholder's interests in the reviewed artefact, and the information that is relevant from the stakeholder's point of view.
- The instructions explain how to read the reviewed document, and how to extract the relevant information from it. The creation of the high-level work product is also specified.
- A set of questions is provided that are answered while following the instructions. The questions focus the attention of the reviewer on specific aspects of the artefact.

3.5 Example

An example of a tester scenario for reviewing requirements specifications is provided in [Basili et al., 1996]. However, the example does not have an explicit introduction part. An interpolated and slightly modified version of that example is provided in Figure 3.



A scenario for a tester's perspective

Introduction

Assume you are reviewing the requirements specification from the perspective of a tester. The tester makes sure that the requirements are correct by creating a set of test cases that covers all relevant functionality of the system. A tester needs requirements that are testable and unambiguous. In this perspective you will create test cases for requirements. A test case consists of inputs, and the expected outputs for these inputs. Follow the instructions and answer the questions.

Instructions

For each requirement, generate a test or set of tests that allow you to ensure that an implementation of the system satisfies the requirement. Use your standard test approach and technique, and incorporate test criteria in the test suite. In doing so, ask your self the questions provided below. Write down all identified defects.

Questions

1. Do you have all the information necessary to identify the item being tested and the test criteria? Can you generate a reasonable test case for each item based upon the criteria?
2. Can you be sure that the tests generated will yield the correct values in the correct units?
3. Are there other interpretations of this requirement that the implementer might make based upon the way the requirement is defined? Will this affect the tests you generate?
4. Is there another requirement for which you would generate a similar test case but would get a contradictory result?
5. Does the requirement make sense from what you know about the application or From what is specified in the general description?

Figure 3. An example of a scenario for reviewing requirements specifications from the tester's perspective.

3.6 Scenario development

The perspective-based reading technique does not include the scenarios themselves that a reviewer could use. It is possible that the reviewed software artefact is similar to the artefacts examined in research papers. In these cases, the examples provided by the researchers (e.g. [Basili, 2011]) can be used as a foundation for the scenarios.

If the reviewed software artefact differs in a significant way from these examples, new scenarios must be created. A process for developing new PBR scenarios is introduced in [Laitenberger and Atkinson, 1999]. The process consists of five steps:

1. **Identification of review documents**. As the inspected software artefact (e.g. a requirements specification of a particular subsystem) has been determined, the documents containing relevant information about that system need to be identified and gathered. The documents can be textual descriptions, design documents, or graphical models.
2. **Stakeholder identification**. The stakeholders that have a particular role in the software development process are specified. Possible roles include: the producer of the preceding description of the artefact, the producer of the subsequent description of the artefact, a tester, a maintainer, a user, and a domain expert. The most relevant stakeholders should be selected as the perspectives for the scenarios.
3. **Identification of relevant information**. The most important information for each perspective is identified. The stakeholders can be interviewed to get answers to questions such as: What does the particular stakeholder need to know about the document to complete their task? How is this information extracted from the document?
4. **Creating scenario instructions**. Now the scenario can be written. The introductory part of the scenario describes the interests of a stakeholder. The instructions guide the reviewer to extract relevant information, as identified in the previous step. The instructions should be written in a detailed manner. The instructions should also demand that the inspector documents the work.
5. **Creating scenario questions.** Questions should be written based on typical problems in the particular environment. The questions should be such that they can be answered with the understanding achieved based on the extracted information. The questions should take into account all relevant defect types. As an example, requirements defects taxonomy was created in [Shull et al., 2000] in order to facilitate question composition so that all defect types are taken into consideration. The taxonomy is presented in Table 1.

Once the scenarios have been established, they can be used on all documents of the same type. In practice, as the scenarios have been used in reviews, the scenarios should, if necessary, be modified and improved based on the experience of applying them.

Table 1. Defect types in requirements.

Missing information	Any significant requirement related to functionality, performance, design constraints, attributes, or external interface that is not included. Undefined software responses to all realisable classes of input data in all realisable classes of situations. Sections of the requirements document. Figure labels and references, tables, and diagrams. Definitions of terms and units of measurement.
Ambiguous information	Multiple interpretations caused by using multiple terms for the same characteristic or multiple meanings of a term in a particular context.
Inconsistent information	Two or more requirements that conflict with one another.
Incorrect fact	A requirement-asserted fact that cannot be true under the conditions specified for the system.
Extraneous information	Unnecessary or unused information (at best, it is irrelevant; at worst, it may confuse requirements users).
Miscellaneous defects	Other errors, such as including a requirement in the wrong section.

3.7 Review of empirical research on perspective-based reading

PBR has already been studied empirically in several different studies. Some results are summarised below.

Results indicating that PBR is more effective than an ad-hoc or a checklist-based technique:

1. In [Basili et al., 1996], two runs of a controlled experiment were conducted to test the effectiveness of PBR against a reading technique usually used at NASA. Specifically, the hypothesis was that PBR would provide a wider coverage of detected defects than the ad-hoc technique as a result of the non-overlapping inspector perspectives. The reading techniques were applied to requirements documents. Teams applying PBR achieved significantly better coverage of documents than teams that did not apply PBR. In addition, the individual PBR reviewers performed significantly better than ad-hoc reviewers on generic requirements documents in the second controlled experiment.

2. The PBR technique was studied in a fractional factorial experiment to examine the technique's effectiveness in supporting individual defect detection in code documents. The results indicated that PBR has an influence on individual defect detection, and that the overlap of defects detected from different perspectives is low. The experiment also indicated that defect detection is more of an individual than a group activity, suggesting that smaller inspection teams would be more effective [Laitenberger and DeBaud, 1997].
3. PBR was compared to checklist-based reading for defect detection in code documents. The comparison was a series of three studies: a quasi-experiment and two internal replications. PBR was statistically found to be more effective than CBR. In addition, the PBR was more cost-effective than CBR [Laitenberger et al., 2000].
4. [Laitenberger, 2001] reviews the results of three empirical studies that compare the cost-effectiveness of PBR, checklist-based reading and ad-hoc reading. Results indicated that PBR is a more cost-effective technique, and that the effect of human experience on inspection results is reduced when PBR is used.
5. An experimental evaluation of the PBR technique determined that the technique can improve the number of defects found by both individuals and teams in certain contexts. The PBR technique was found most useful for novice users. More experienced reviewers tend to fall back into using traditional techniques when reviewing documents, as their previous experience interferes with the process of the reading technique [Shull, 1998].

Results indicating that PBR is not significantly more effective than ad-hoc reading or checklist-based reading:

1. A study of PBR [Basili et al., 1996] was replicated in order to better understand the complementary aspects of different PBR perspectives. The results showed that PBR was more effective than CBR in one of the two requirements documents used in the comparison. The study also found that when the perspectives found similar defects, there was no overall benefit observed for the perspective-based technique [Maldonado et al., 2006].
2. [Freimut et al., 2001] examined the effect of reading techniques on the accuracy of various defect content estimation techniques (estimating the total number of defects in a document). The empirical study compared two reading techniques: checklist-based reading and scenario-based reading (which was a combination of both perspective-based reading and traceability-based reading). The inspected artefact was a requirements document. No difference in the accuracy was observed between the two reading techniques.

Five of the research papers report that PBR has been significantly more effective than a more traditional reading technique. Two results indicate that the difference between the techniques' effectiveness is not statistically significant.

4. Application of the PBR technique in the nuclear domain

To apply perspective-based reading in the nuclear domain, one should first decide what software artefacts are under review. It has already been demonstrated that PBR is applicable to various software artefacts such as requirements documents [Basili et al., 1996], design models [Laitenberger and Atkinson, 1999], and code documents [Laitenberger and DeBaud, 1997]. The nuclear domain software life cycle and the software artefacts do not much differ from the generic software artefacts. As an example, the nuclear domain Category A software standard IEC 60880 [IEC 60880, 2006] specifies the following documents in its typical list of software documentation:

- System requirements specification
- System specification
- Software requirements specification
- Software quality assurance plan
- Detailed recommendations
- Software verification plan
- Software aspects of system integration plan
- Software design specification
- Software design verification report
- Software test specification
- Software code verification report
- Software test report
- Software aspects of integrated system verification report
- Software aspects of the system validation plan
- Software aspects of the system validation report
- Software user manual
- Software aspects of the commissioning test plan
- Software aspects of the commissioning test report
- Documents relating to software modifications
- Anomaly report
- Software modification request
- Software modification report
- Software modification control history.

None of the documents in the list is nuclear specific, but the documents are likely to contain aspects only related to the nuclear domain. Therefore, it seems that the PBR technique can be used at least on the main nuclear domain software artefacts, such as requirements specification, design phase documents, and code. Other documentation from the list above is also a potential target for PBR reviews. For each different software artefact planned for PBR application, several detailed PBR scenarios should first be developed according to the guidelines provided in Section 3.6.

In what follows, example PBR scenarios are developed for nuclear-domain requirements specifications. An exemplar requirements specification [EPRI, 2000] is used as a reference for a typical requirements specification document. The resulting scenarios are in Appendix A.

4.1 Example system description and requirements

The first step of PBR scenario development is to identify the documents relevant to the software artefact under review. In this running example, the target system is a rod control system upgrade. The overall system and the generic requirements specification for the system are described in [EPRI, 2000]. The document is a generic requirements specification for a rod control system upgrade for Westinghouse pressurized water reactors. Similar requirements specifications could be used in, for example, nuclear automation renewal projects.

4.1.1 Rod control system

The rod control system is part of the power control system. The power control system adjusts reactor power and temperature deviations through control rod motion. The task of the rod control system is to move the control rods based on the demand signals from the reactor control system or the reactor operator. The control rods are organised into groups, and these groups are further organised into shutdown banks and control banks. The control bank groups are used in reactor power control. All shutdown and control groups are used to safely shut down the reactor.

The rod control system and other interacting systems and components are depicted in Figure 4. The functions of the rod control system can be divided into three modules:

- Rod control logic: contains the timing logic, bank sequencing, bank overlap, and interlock processing functions. Based on signals from the operator and the reactor control function, command signals are output to power cabinets to move the control rods.
- Power cabinets: Provides electricity to the system.
- Reactor Operator HMI: The user interface that uses “soft controls” and CRT displays.

The rod control system interacts with other systems, including:

- Control rod drive mechanisms: An electromagnetic device which sequentially energises and de-energises three coils to move a rod control cluster up or down in single step increments.
- Rod control motor-generator sets: The source of electrical power to the control rod drive mechanisms and the rod control system is from two motor-generator sets.
- Reactor trip breakers: The reactor trip breakers can be tripped manually by operators or automatically in the reactor protection system. A trip causes all control bank and shutdown bank rods to drop into the core.
- Rod stop interlocks: Prevent uncontrolled power escalation by blocking rod withdrawal while allowing rod insertion.
- Rod position indication system: Provides axial position information to the main control room for all of the control rod clusters.
- Reactor control system (process instrumentation such as reactor coolant temperatures, NIS power, turbine power, etc.)
- Plant computer software: Provides the operator with comparative information regarding rod positions.

The upgrade includes replacing logic and power cabinets with modern hardware and software. A CRT-based human-system interface is specified.

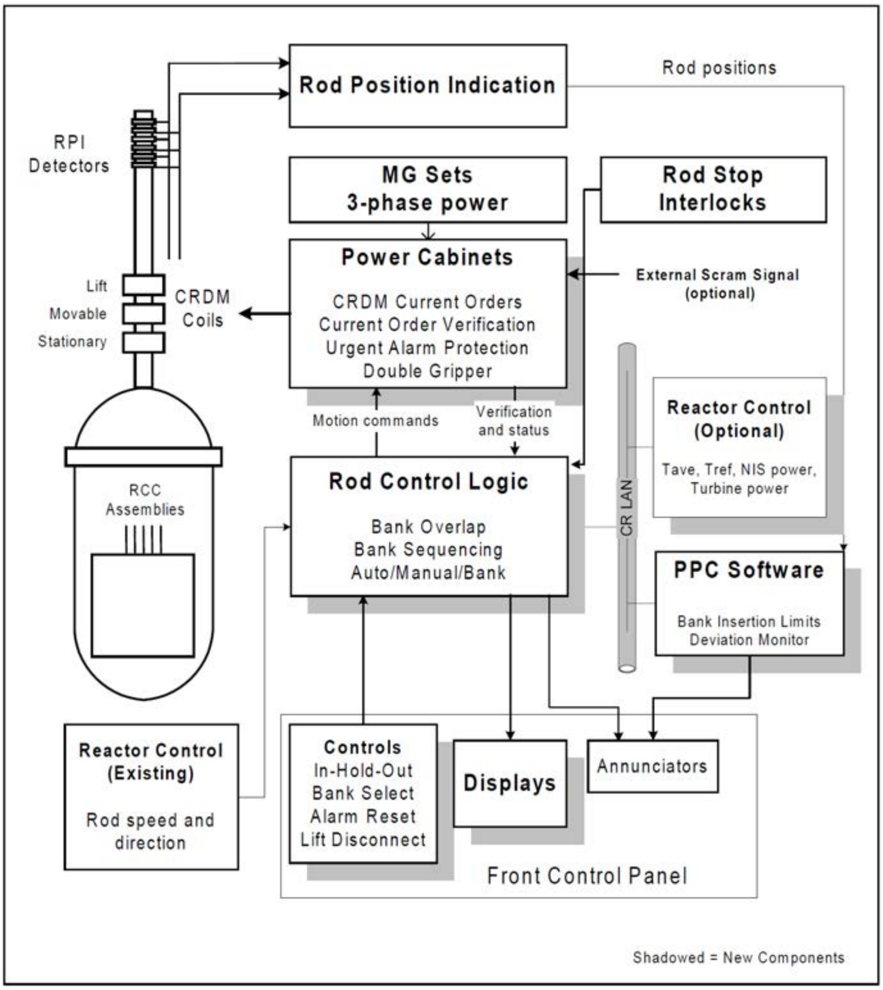


Figure 4. The rod control system upgrade.

4.1.2 EPRI requirements specification

The scope of the document covers hardware, software, and system requirements. Interface requirements, power generation, the user interface, and the local area network are also covered to some extent. The EPRI document is quite generic and can be adapted to specific plants. In this work, the main focus is on the software requirements of the EPRI document.

4.2 Identification of document stakeholders

The next step in scenario development is the identification of document stakeholders. Nuclear domain requirements specifications are generally developed similarly to any other requirements specifications. The stakeholders for generic software requirements specifications are identified for example in [Shull et al., 2000]. Stakeholder identification requires the identification of who will use the system and requirements specification and in what way. In [Shull et al., 2000], it is identified that a requirements specification has three major uses at later stages of the life cycle:

- A description of the customer's needs
- A basis for the system design
- A point of comparison for system tests.

This suggests that three distinct perspectives are relevant for generic requirements specifications: a user perspective, a designer perspective, and a tester perspective. The designer requires that sufficient detail is provided in the requirements specification. The tester wants requirements that can be tested. A user of the system requires general completeness and correctness.

These three perspectives are also relevant in nuclear domain reviews. However, other perspectives might be relevant as well. The life cycle of a requirements specification should be analysed to find other potentially relevant perspectives. Special features of nuclear-domain systems, such as emphasised safety requirements, the regulator's role, and the long life-time of the plant, should also influence the stakeholder identification process. Nuclear-specific aspects related to requirements specifications are discussed in what follows.

4.2.1 Long system life-time

A nuclear power plant (NPP) has quite a long operational life-time (appr. 60 years). Renewal of systems within this period is often necessary. An existing system may have to be replaced or entirely new systems built. Reasons for system renewal might include: new regulation demanding better systems, old systems becoming obsolete, or an increase in the safety of the plant. From the system renewal perspective, it is relevant that the requirements written for a system are easily extensible. Furthermore, it is relevant that the reasoning behind the written requirements is stated explicitly so that the requirements are not later casually changed or neglected.

The long life-span of an NPP also indicates the importance of the repairability of the system, the modifiability of the system, and the importance of periodic tests and overall maintenance of the system. From this perspective it is important that the requirements pay attention to, for example, defect detection and correction methods.

The concept of maintainability [Avizienis et al., 2004] covers all these issues related to a long system life-span. Maintainability is the ability to undergo modifications and repairs, including:

- Corrective maintenance (repairs)
- Preventive maintenance (removal of dormant faults)
- Adaptive maintenance (modifications caused by environment changes)
- Augmentative maintenance (changes to the system's function).

The maintainability of an NPP system is seen as an important aspect. In Appendix A, we define a maintainer's perspective scenario for the inspection of requirements specifications. This perspective encompasses the concern for proper system renewal, modification, and repairs.

4.2.2 System design process complexity

The design process of an NPP-related system is a complex process. Many things have to be taken into consideration. Typically, systems are designed by many people together, responsible for different areas of the design. One person hardly has all the necessary knowledge and skills to evaluate all the relevant design aspects of a system at the same time. In addition to the typical designer role, a nuclear-domain system may have, for instance, the following roles related to system design:

- System safety engineer, making sure that the system remains operational even as individual components fail
- Plant process designer, focusing on the nuclear process
- Electrical wiring designer
- Nuclear physicist
- HMI designer.

The special aspect here is the large amount of information needed to design a nuclear-specific system. There are many design constraints and rules that you have to know in order to analyse whether the requirements are complete. In an NPP, it is not enough that the system functions correctly; it also has to follow, for example, licensing regulations, plant procedures, and standards.

Each of the various designer roles can be seen as a potential PBR perspective. However, that leads to a large group of inspectors. In our approach, the designer perspective defined in Appendix A corresponds to all these designer roles. It is important from the designer's perspective that all design constraints related to the different aspects above are explicitly written down in the requirements specification.

The previous PBR techniques have neglected many of the issues related to the design of a complex system (such as an NPP safety system). This is because the systems reviewed using PBR have been software implementations and the related hardware and environment issues have not been part of the inspection. Design constraints have not been addressed as a specific concern. In our technique, evaluating design constraint coverage is seen as an important task. The task is interpreted as a part of the designer's perspective.

4.2.3 Regulator's role

One special aspect of the nuclear-domain requirements is that they have to be approved by the regulator. In practice, this means that the requirements specification is reviewed by the regulator body. The objective of the regulator's review is to compare the requirements to the requirements given by law and by the Finnish YVL guides. The regulator wants to make sure that the main safety requirements and design principles are followed and that reactor safety is ensured. Other than that, the regulator probably focuses on the referred standards followed, and the overall quality of the documentation.

4.2.4 Safety importance

Another nuclear aspect is the great importance of safety. The "safety as high as reasonably achievable" principle is often quoted. The nuclear-domain safety precautions are manifold: the defence-in-depth principle calls for multiple protection measures that back each other up in case one measure fails. In addition, some protection measures are built in as redundant and diverse to achieve failure tolerance.

In spite of all these precautions, the effects of a faulty safety system can still be severe, and thus a lot of effort is put into the verification of the functions of the systems. For example, a nuclear safety system is typically thoroughly tested and simulated, and also undergoes various independent analyses, including analysis by formal and semi-formal analysis methods. From the safety point of view, it is essential that the verification is extensive and the behaviour in all improbable corner cases of the system function are also analysed. It is also imperative that the potential common-cause failures are thoroughly evaluated and identified.

An independent verifier is also a stakeholder in the requirements specification. The verifier performs, for example, a formal analysis against the requirements specification. Similarly to the tester, the verifier needs requirements that can be verified. When formal methods are used, it is also necessary that the requirements are detailed enough to consider all possible system behaviour. A verifier might also be interested in looking for potential loopholes in the requirements.

4.2.5 Contract work chains

Construction of a nuclear power plant is a huge project. Some work is distributed to contractors. For example, the requirements specification might be given as input to a contractor to create some part of the system. The contractor might hire a subcontractor to perform the work. The requirements specification is given to the subcontractor. The potential problem is that the subcontractor might not be aware of all related nuclear-specific guidelines and design principles that are not explicitly mentioned in the requirements specification documentation. Considering this, it might be interesting to find out whether the individual requirements are such that they could have varying interpretations depending on the background of the reader. Could a person with no detailed nuclear-domain knowledge understand the requirements in a different way?

4.2.6 PBR perspective identification and specification

Based on the considerations above, we identify seven possible stakeholder roles that can be used as a perspective in the PBR reading technique. The specifics (relevant information for the perspective, a work product created to obtain this information) of each perspective are discussed below. The resulting reading scenarios are in Appendix A.

- **Designer**: Reads the requirements from the system designer’s perspective. Additionally, must evaluate whether the various design constraints related to the system are explicitly identified in the requirements. The designer wants to know whether the requirements and design constraints are detailed enough for the design to be made. This information can be easily found by creating a small prototype design (for example a state machine diagram) and checking if some aspects of the design are unclear.
- **Tester**: Reads the requirements from the tester’s perspective. Is interested in whether the requirements can be used to form test cases. The tester samples the requirements and writes actual test cases to find out if the requirements are suitable.
- **User/operator**: Reads the requirements from the system user’s perspective. Is interested in the overall completeness and correctness of the requirements. The user creates a set of the most common use cases based on the requirements, and examines whether any difficulties in the use-case specification arise.
- **Maintainer**: Focuses on maintainability issues. Are the requirements such that they support system repairs and modifications? To find this out, the maintainer produces a data flow diagram describing the system functions, and analyses the dependencies between these functions. For each identified function, the maintainer writes down the motivation behind this function. The maintainer also identifies other requirements that are dependent on changes related to the function. The maintainer writes down how defects in the function are detected and corrected.
- **Verifier**: Focuses on corner case analyses and identification of common-cause failures in the requirements. In order to identify whether unexpected

corner cases are addressed, the verifier samples the requirements and produces a set of negative requirements (requiring what the system should not do). The verifier also produces a fault tree of the system and identifies: (1) where similar functions are used in many system components, (2) system functions that are dependent on hardware located in the same physical space, (3) probable components that will require maintenance. (Note: the verifier is typically also interested in the testability of the requirements, but because that aspect is already handled by the tester perspective, the verifier shall focus on other matters.)

- **Regulator**: Focuses on overall quality and references, and compares the requirements specification to other regulations. The regulator creates a list of standards and other relevant documents that are referred to in the requirements specification. The regulator produces a data flow diagram of the system components. Based on the diagram, the regulator identifies the parts in which the concepts of redundancy, diversity, and physical separation are applied. The regulator also writes down inconsistencies within the document, and compares the document structure to an example structure provided in a standard (e.g. IEEE 830-1998).
- **Contractor**: Focuses on whether the requirements can be misunderstood if they are read by a person not familiar with the nuclear domain. The contractor samples the requirements and writes down whether nuclear-domain knowledge is required to understand the requirement. Is this knowledge referred to in the proximity of the requirement? What would be the possible alternative interpretation of the requirement?

4.3 Scenario usage considerations

Our analysis led to a total of seven separate scenarios. Using all of the scenarios would require an inspection group consisting of at least seven people. This is quite a lot, and many researchers suggest that using smaller groups is more effective. We also recommend that discretion is used when selecting PBR perspectives to be used in an inspection. We have defined a set of perspectives that, in general, have greater relevance in a nuclear-domain system. However, only the most relevant perspectives should be selected, depending on the case, so that the effectiveness of the inspection does not suffer.

In addition, the defined perspectives and scenarios are not of equal importance. The most fundamental perspectives, such as designer and tester, should perhaps always be included. The contractor perspective, on the other hand, is quite one-sided and not necessarily important at all. It might also be possible to merge some of the tasks given for the scenarios into a single scenario, thus decreasing the number of scenarios but maintaining the same tasks.

It would be important for the usability and effectiveness of the defined scenarios to be evaluated in a real case study. It is difficult to estimate their usefulness without experimentation. PBR inspection papers also encourage refining and modifying the scenarios based on the experiences of using them. Only actual use reveals which scenarios are useful and which ones should be discarded in future reviews.

The scenarios should be used by people who have previous experience in the context of the perspective. The best review results have been achieved in cases where the reviewers get to make use of their special know-how. Reviewer experience has a lot of importance when reviewers using the designer perspective are selected, because the perspective requires a lot of knowledge from various domains. It might be useful to use several reviewers to apply the designer's perspective, each with a slightly different background (for example process engineer, safety engineer, and physicist).

5. Summary

Inspections and reviews are one of the most effective ways of detecting errors in software development. The methods are also cost-effective because the defects can be spotted early in the development, and the cost of repairing the defects is thus lower.

Reading techniques are procedures that are used in the inspection and review of software artefacts. The most common reading techniques are simple ad-hoc reading and checklist-based reading. However, more advanced and detailed procedures exist for various purposes.

In this paper, state-of-the-art reading techniques used in software inspections have been reviewed. The study identified 13 reading techniques that can be used in the inspection of a software product. Some reading techniques are specific to a life-cycle phase-, some reading techniques can be used on practically any software artefact. Some empirical research concerning the reading techniques was also reviewed. The majority of the results indicate that advanced reading techniques such as perspective-based reading can find more defects and are more cost-effective than ad-hoc reading and checklist-based reading.

The perspective-based reading technique was examined more closely. The reading technique is based on reading a software document from different stakeholder perspectives. Detailed instructions called scenarios are written for each perspective. The idea is that inspectors detect fewer overlapping defects and are more effective when they focus only on a subset of the matter. In PBR, inspectors are also instructed to create their own work products, in order to force the inspectors to think from the given perspectives.

The application of the PBR technique was assessed in the nuclear context. The technique seems applicable to the inspection of several nuclear-domain documents. As an example of how the technique could be used, PBR was applied to the inspection of requirements specifications. The EPRI requirements specification [EPRI, 2000] was used as an example of a typical requirements specification in the nuclear field. In this context, a few nuclear specific aspects were identified that should be considered: the complexity of the design process, the long system lifespan, the regulator’s role in system development, sub-contractor issues, and the great importance of safety.

Based on these special concerns, seven PBR perspectives were identified. A reading scenario was written for each perspective. The derived reading perspectives/scenarios were: designer, tester, user/operator, maintainer, verifier, regulator, and contractor. The scenarios are intended to be used by a separate inspector so that each inspector focuses on only a small part of the requirements specification. If a smaller group of inspectors should be available, only the most relevant perspectives should be selected, depending on the case, so that the effectiveness of the inspection does not suffer. Not all defined perspectives and scenarios are of equal importance. The most fundamental perspectives, such as designer and tester, should perhaps always be included in the inspection. A contractor perspective should carry a smaller weight than these. Finally, the best review results are expected when the reviewers are able to make use of their special know-how. Thus, the person reviewing from the designer's perspective should have some previous knowledge related to the design phase.

The effectiveness of the created PBR scenarios should be determined (and the scenarios modified based on the experiences) in an actual case study. This is left for future research. The created scenarios are intended for requirements specification reviews only. If documents in other life-cycle phases are reviewed, the scenarios should be changed to reflect the stakeholders of those types of documents. The scenario development process for that purpose is reviewed in this report.

References

Avizienis, A., Laprie, J.-C., Randell, B. and Landwehr, C. 2004. Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing, Vol. 1, pp. 11–33.

Barnard, J. and Price, A. 1994. Managing Code Inspection Information. IEEE Software, Vol. 11, Issue 2, pp. 59–69.

Basili, V. R., Green, S., Laitenberger, O., Lanubile, F., Shull, F., Sørumgård, S., and Zelkowitz, M. 2011. Lab package for the empirical investigation of perspective-based reading. http://www.cs.umd.edu/projects/SoftEng/ESEG/manual/pbr_package/manual.html (accessed: 15 March, 2012).

Basili, V. R., Green, S., Laitenberger, O., Shull, F., Sørumgård, S. and Zelkowitz, M. V. 1996. The empirical investigation of perspective-based reading. Empir Softw Eng-Int J., Vol. 1, pp. 133–164.

Cheng, B. and Jeffrey, R. 1996. Comparing inspection strategies for software requirements specifications. In: Proceedings of the 1996 Australian Software Engineering Conference, pp. 203–211.

Christenson, D.A., Huang, S.T. and Lamperez, A.J. 1990. Statistical quality control applied to code inspections. IEEE Journal of Selected Areas of Communication, Vol. 8, Issue 2, pp. 196–200.

Dunsmore, A., Roper, M. and Wood, M. 2001. Systematic object-oriented inspection – An empirical study. In: Proceedings of ICSE ’01 23rd International Conference on Software Engineering. IEEE Computer Society Washington DC, USA.

Dyer, M. 1992. Verification-based Inspection. In: Proceedings of the 26th Annual Hawaii International Conference on System Sciences, pp. 418–427.

EPRI 2000. Requirements Specification for Rod Control System Upgrade: A Generic Specification for Westinghouse Pressurized Water Reactors. Electric Power Research Institute 1000969.

Fagan, M. E. 1976. Design and code inspections to reduce errors in program development. IBM Sys. J. Vol. 15, No. 3, pp. 182–211.

Fagan, M. E. 1986. Advances in software inspections. IEEE Transactions on Software Engineering, Vol. 12, Issue 7, pp. 744–751.

Freimut, B., Laitenberger, O. and Biffl, S. 2001. Investigating the impact of reading techniques on the accuracy of different defect content estimation techniques. In: Proceedings of the 7th International Symposium on Software Metrics (METRICS '01). IEEE Computer Society, Washington DC, USA.

Grady, R. B. and van Slack, T. 1994. Key lessons in achieving widespread inspection use. IEEE Software, Vol. 11, Issue 4, pp. 46–57.

IEC 2006. Nuclear power plants – Instrumentation and control systems important to safety – Software aspects for computer-based systems performing category A functions, International Electrotechnical Commission 60880.

IEEE Standard 2008. IEEE Standard for Software Reviews and Audits, 1028-2008.

Kan, S. H. 1995. Metrics and models in software quality engineering. Addison-Wesley Publishing Company.

Kaner, C. 1998. The performance of the n-fold requirement inspection method. Requirements Engineering Journal, Vol. 2, No. 2, pp. 114–116.

Kelly, D. and Shepard, T. Task-directed software inspection. 2004. J. Syst. Softw., Vol. 73, Issue 2, pp. 361–368.

Kelly, J. C., Sherif, J. S. and Hops, J. 1992. An analysis of defect densities found during software inspections. Journal of Systems and Software, Vol. 17, Issue 2, pp. 111–117.

Laitenberger, O. 2001. Cost-effective Detection of Software Defects through Perspective-based Inspections. Empirical Software Engineering, Vol. 6, Issue 1, pp. 81–84.

Laitenberger, O. 2002. A survey of software inspection technologies. In: Handbook on Software Engineering and Knowledge Engineering. Vol. 2: Emerging Technologies, River Edge: World Scientific, pp. 517–555.

Laitenberger, O. and Atkinson, C. 1999. Generalizing perspective-based inspection to handle object-oriented development artifacts. In: Proceedings of the 1999 International Conference on Software Engineering, 1999. Los Angeles, CA, USA, pp. 494–503.

Laitenberger, O., Beil, T. and Schwinn, T. 2002. An industrial case study to examine a non-traditional inspection implementation for requirements specifications. Empirical Software Engineering, Vol. 7, Issue 4, pp. 345–374.

Laitenberger, O. and DeBaud, J.-M. 1997. Perspective-based reading of code documents at Robert Bosch GmbH, Tech. Rep. ISERN-97-14.

Laitenberger, O., El Emam, K. and Harbich, T. 2000. An internally replicated quasi-experimental comparison of checklist and perspective-based reading of code documents. IEEE Transactions on Software Engineering.

Mafra, S. N. and Travassos, G. H. 2005. A skeptical discussion regarding experimentation in software engineering based on reading techniques studies. http://lens.cos.ufrj.br:8080/eselaw/proceedings/2005/interestedareas/ESELAW2005_paper10_mafra.pdf (accessed: 15 March, 2012).

Maldonado, J. C., Carver, J., Shull, F., Fabbri, S., Dória, E., Martimiano, L., Mendonça, M. and Basili, V. 2006. Perspective-based reading: A replicated experiment focused on individual reviewer effectiveness. Empirical Software Engineering, Vol. 11, Issue 1, pp. 119–142.

Parnas, D. L. and Weiss, D. 1985. Active design reviews: principles and practices. In: Proceedings of the 8th International Conference on Software Engineering, pp. 132–136. Also Available as NRL Report 8927, 18 November 1985.

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, Vol. 23, Issue 3, pp. 129–144.

Porter, A. and Votta, L. G. 1994. An experiment to assess different defect detection methods for software requirements inspections. In: Proceedings of the 16th International Conference on Software Engineering, pp. 103–112.

Porter, A. A. and Votta, L. G. 1997. What makes inspections work? IEEE Software, Vol. 14, Issue 6, pp. 99–102.

Porter, A. and Votta, L. 1998. Comparing detection methods for software requirements inspections: A replication using professional subjects. Empirical Software Engineering, Vol. 3, Issue 4, pp. 355–379.

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, Vol. 21, Issue 6, pp. 563–575.

Remus, H. 1984. Integrated software validation in the view of inspections/reviews. In: Proc. of a symposium on Software validation: inspection-testing-verification-alternatives. Elsevier North-Holland, Inc. New York, NY, USA ISBN 0-444-87593-X, pp. 57–64.

Rifkin, S. and Deimel, L. 1994. Applying program comprehension techniques to improve software inspection. In: Proceedings of the 19th Annual NASA Software Eng. Laboratory Workshop. NASA.

Robbins, B. and Carver, J. 2009. Cognitive factors in perspective-based reading: A protocol analysis study. In: Proceedings of the 2009 International Symposium on Empirical Software Engineering and Measurement (ESEM), Lake Buena Vista, FL, USA, October 15–16, 2009.

Saaty, T. L. and Vargas, L. G. 2001. Models, methods, concepts & applications of the analytic hierarchy process. Kluwer Academic, Boston, USA.

Shull, F. 1998. Developing techniques for using software documents: A series of empirical studies. Ph.D. Dissertation. University of Maryland at College Park. MD, USA.

Shull, F., Rus, I. and Basili, V. 2000. How perspective-based reading can improve requirements inspections. Computer, Vol. 33, Issue 7, pp. 73–79.

Selby, R. W., Basili, V. R. and Baker, F. T. 1987. Cleanroom software development: An empirical evaluation. IEEE Transactions on Software Engineering, SE-13, Issue 9, pp. 1027–1037.

Thelin T., Runeson, P. and Regnell, B. 2001. Usage-based reading – an experiment to guide reviewers with use cases. Information and Software Technology, Vol. 43, Issue 15, pp. 925–938.

Thelin, T., Runeson, P. and Wohlin, C. 2003. An experimental comparison of usage-based and checklist-based reading. IEEE Transactions on Software Engineering, Vol. 29, Issue 8, pp. 687–704.

Travassos, G. H., Shull, F., Carver, J. and Basili, V. R. 1999a. Reading techniques for OO design inspections. In: Proceedings of the Twenty-fourth Annual Software Engineering Workshop. Goddard Space Flight Center, Greenbelt, MD, USA, December 1999.

Travassos, G. H., Shull, F., Fredericks, M. and Basili, V. R. 1999b. Detecting defects in object oriented designs: Using reading techniques to increase software quality. In: Conference on Object-oriented Programming Systems, Languages & Applications, pp. 44–56.

Weller, E. F. 1993. Lessons from three years of inspection data. IEEE Software, Vol. 10, Issue 5, pp. 38–45.

Zhang, Z., Basili, V. and Shneiderman, B. 1998. An empirical study of perspective-based usability inspection. In: Proceedings of the Human Factors and Ergonomics Society 42nd Annual Meeting. Chicago, IL, USA, pp. 1346–1350.

Appendix A: Example scenarios

A.1 Designer scenario for requirements specifications

Introduction: Assume you are reading the requirements from a system designer's perspective. You are interested in knowing whether the requirements are detailed enough for the design to be made based on them. Concentrate also on whether the design constraints cover all relevant aspects of the system.
First read the instructions and questions below and then proceed to follow the instructions while writing down observations related to the questions. Document all your work products so that the inspection is traceable and your work is repeatable.

Instructions: Based on the requirements, create a high-level design of the system that encompasses all system functions, relevant data objects and structures. As a part of your design, produce the data flow diagram of the system.

Questions:

1. Are the system interfaces appropriately defined?
2. Is all necessary functionality defined in the requirements?
3. Is all information available to create the design?
4. Are there unclear requirements?
5. Are all relevant design constraints of the system explained?
6. Are all necessary data sources, destinations and stores defined? Are data types defined?
7. Can the data flow diagram be used to trace the behaviour in actual cases?

A.2 Tester scenario for requirements specifications

Introduction: Assume you are reading the requirements from a system tester's perspective. You are interested in knowing whether the requirements are such that test cases can be written based on them. Concentrate on finding vague, ambiguous and unclear requirements.

First read the instructions and questions below and then proceed to follow the instructions while writing down observations related to the questions. Document all your work products so that the inspection is traceable and your work is repeatable.

Instructions: For each requirement, create a test case or a set of test cases that can be used to verify that requirement. Try to create tests based on the equivalence sets of inputs (one test case for each functionally different outcome). Document inputs and expected outputs for each test case.

Questions:

1. Is all relevant information available to produce the test cases (inputs)?
2. Is the outcome of the test specified unambiguously?
3. Is there an alternative interpretation of the requirement that would result in a functionally different outcome?

A.3 User/operator scenario for requirements specifications

Introduction: Assume you are reading the requirements from the operator's perspective. You are interested in knowing whether the requirements adequately describe the common functions of the system. Concentrate on the overall correctness and completeness of the requirements.
First read the instructions and questions below and then proceed to follow the instructions while writing down observations related to the questions. Document all your work products so that the inspection is traceable and your work is repeatable.

Instructions: Based on the requirements, produce a set of the most common and critical use cases. Document the use cases carefully. Define the inputs required to perform the user functions and the outputs generated by each function. Define the flow of system control in a diagram.

Questions:

1. Is all relevant information available for the creation of the use cases?
2. Are the requirements ambiguous?
3. Are there situations in which the wrong operator action causes safety issues? Is the proper operation in these situations evident?

A.4 Maintainer scenario for requirements specifications

Introduction: Assume you are reading the requirements from a system maintainer's perspective. You are interested in knowing whether the requirements support system repairs and modifications. Concentrate on whether the reasons for the requirements are explained and if there are dependencies between requirements. Low coupling and high cohesion are also indicators of good maintainability of the requirements.

First read the instructions and questions below and then proceed to follow the instructions while writing down observations related to the questions. Document all your work products so that the inspection is traceable and your work is repeatable.

Instructions: Based on the requirements, create a data flow diagram describing the functions of the system. Analyse the dependencies between these functions. For each function:

1. Write down the motivation behind the requirements for the function.
2. Write down the defect detection and correction methods related to that function.

Questions:

1. Is maintenance/repair of the system parts acknowledged in the requirements?
2. Are the requirements written in a coherent manner, using explicit cross-referencing?
3. Is there redundancy in the requirements specification (with the same requirement mentioned in several parts of the document)?
4. Is each requirement expressed separately, using a unique name or reference number?
5. If possible, is there a reference to the source of the requirement in earlier documents?
6. Could the system use more modularity in its structure?
7. Could the interaction between different parts of the system be decreased?

A.5 Verifier scenario for requirements specifications

Introduction: Assume you are reading the requirements from the perspective of an independent verifier using formal methods. You are interested in knowing whether the requirements cover all possible behaviour of the system. Concentrate on improbable cases, unusual user actions, and so on. Try to think of scenarios in which contradictions could arise. In addition, try to think of scenarios in which a single event could cause the system to fail (common-cause failures).

First read the instructions and questions below and then proceed to follow the instructions while writing down observations related to the questions. Document all your work products so that the inspection is traceable and your work is repeatable.

Instructions:

1. For each requirement, think of a possible negative version of that requirement (requiring what the system should not do). Write down the negative counter-parts.
2. Create a high-level fault tree of the system. Identify the minimal cut sets of the fault tree. Identify and document parts of the system: (1) where similar functions are used in many system components, (2) where system functions that are dependent on hardware are located in the same physical space, (3) where components will often require maintenance.

Questions:

1. Is the written negative requirement also a requirement for the system? Has it been explicitly mentioned in the requirement specification? Should it be mentioned?
2. Do the negative requirements remind you of cases in which the system functionality or proper action is not specified?
3. Is there functionality in the system that might contradict the written negative requirement? Can it be verified that the system does not necessarily perform disallowed actions?
4. Could the system fail because of a functionally similar design being used in many components?
5. Is the system adequately distributed so that an environmental disaster does not affect all critical system components at the same time?
6. Are there components in which the wrong human action (e.g. maintenance) could cause the failure of the system?

A.6 Regulator scenario for requirements specifications

Introduction: Assume you are reading the requirements from a regulator's perspective. You are interested in knowing whether the requirement specification is of good quality and follows all necessary laws, guidelines, and standards. Concentrate on whether redundancy, diversity, and physical separation are applied in the system.

First read the instructions and questions below and then proceed to follow the instructions while writing down observations related to the questions. Document all your work products so that the inspection is traceable and your work is repeatable.

Instructions:

1. Based on the requirements, create a data flow diagram depicting the system components. Identify and write down the parts in which the concepts of redundancy, diversity, and physical separation are applied.
2. Create a list of standards and other relevant documents that are referred to in the requirements specification.

Questions:

1. Are the safety principles stated in the Finnish YVL guides applied in an adequate manner?
2. What standards are followed? Are there deficiencies in the references in the document?
3. Are there any inconsistencies within the document?
4. Does the requirements specification address all relevant topics in the example structure of a requirements specification document as required in IEEE 830-1998?
5. Is each requirement expressed separately, using a unique name or reference number?

A.7 Subcontractor scenario for requirements specifications

Introduction: Assume you are reading the requirements from a subcontractor's perspective. You do not have extensive knowledge of specific nuclear-domain design constraints. Concentrate on whether the requirements can be misinterpreted by a person outside the nuclear field.

First read the instructions and questions below and then proceed to follow the instructions while writing down observations related to the questions. Document all your work products so that the inspection is traceable and your work is repeatable.

Instructions: For each requirement, write down the nuclear domain design constraints that have to be known in order to understand it. Think of alternative interpretations and write them down.

Questions:

1. Is the requirement unclear?
2. Could the requirement be written differently?

Is the design constraint related to the requirement mentioned in close proximity to the requirement in the text? Is the design constraint mentioned in the requirements specification at all?

Series title and number
VTT Technology 9

Title	**Application of the perspective-based reading technique in the nuclear I&C context** **CORSICA work report 2011**
Author(s)	Jussi Lahtinen
Abstract	Inspections and reviews are one of the most effective ways of detecting errors in software development. The methods are also cost-effective because defects can be spotted early in the development, and thus the cost of repairing the defects is lower. Reading techniques are the procedures that are used in the inspection or review of a software artefact. The most common procedures are simple ad-hoc reading and a checklist-based reading technique. However, more advanced and detailed procedures have been created for various purposes. This report reviews the state-of-the-art software reading techniques used in inspections and reviews, and briefly reviews some of the empirical research in this context. The majority of the empirical research results indicate that, for example, perspective-based reading is more cost-effective and can detect more defects than more basic reading techniques. This report also describes how perspective-based reading can be applied to the inspection of nuclear-domain requirement specifications. For this purpose, seven perspective-based reading scenarios have been created.
ISBN, ISSN	ISBN 978-951-38-7621-0 (URL: http://www.vtt.fi/publications/index.jsp) ISSN 2242-122X (URL: http://www.vtt.fi/publications/index.jsp)
Date	March 2012
Language	English
Pages	45 p. + app. 7 p.
Name of the project	CORSICA
Commissioned by	
Keywords	software inspection, review, reading technique, perspective-based reading, nuclear, requirements specification
Publisher	VTT Technical Research Centre of Finland P.O. Box 1000, FI-02044 VTT, Finland, Tel. 020 722 111

Application of the perspective-based reading technique in the nuclear I&C context. CORSICA work report 2011

Inspections and reviews are one of the most effective ways of detecting errors in software development. The methods are also cost-effective because defects can be spotted early in the development, and thus the cost of repairing the defects is lower.

Reading techniques are the procedures that are used in the inspection or review of a software artefact. The most common procedures are simple ad-hoc reading and a checklist-based reading technique. However, more advanced and detailed procedures have been created for various purposes.

This report reviews the state-of-the-art software reading techniques used in inspections and reviews, and briefly reviews some of the empirical research in this context. The majority of the empirical research results indicate that, for example, perspective-based reading is more cost-effective and can detect more defects than more basic reading techniques.

This report also describes how perspective-based reading can be applied to the inspection of nuclear-domain requirement specifications. For this purpose, seven perspective-based reading scenarios have been created.

SBN 978-951-38-7621-0 (URL: http://www.vtt.fi/publications/index.jsp)
ISSN 2242-122X (URL: http://www.vtt.fi/publications/index.jsp)