Fraunhofer Institut
Experimentelles
Software Engineering

A Defect Classification Scheme for the Inspection of QUASAR Requirement Documents

Author:
Bernd Freimut
Christian Denger

IESE-Report No. 076.03/E
Version 1.0
September 1, 2003

A publication by Fraunhofer IESE

Fraunhofer IESE is an institute of the Fraunhofer Gesellschaft.
The institute transfers innovative software development techniques, methods and tools into industrial practice, assists companies in building software competencies customized to their needs, and helps them to establish a competetive market position.

Fraunhofer IESE is directed by
Prof. Dr. Dieter Rombach
Sauerwiesen 6
D-67661 Kaiserslautern

Abstract

Software inspection play a crucial role in achieving high quality software right from the beginning. Especially for requirements documents inspections are beneficial as defects can be detected and removed at an early point in time before they can leak into subsequent phases of the development process, where those defects can cause high rework cost and quality problems [Boehm and Basili, 2001].

In order to fully exploit the beneficial aspect of inspections it is necessary to control them: Based on an analysis of the inspection effectiveness (e.g., what type or number of defects have been detected and what type and number might be remaining) it has to be decided whether development can proceed as planned or controlling actions, such as re-inspections, have to be performed.

Defect classification is a technique that can support this activity. Using appropriate defect classification schemes it is possible to get a quantitative statement of the inspection effectiveness (i.e., what type of defects have been found). Examples in the literature demonstrated beneficial applications but are hard to adopt without tailoring the underlying defect classification scheme to the environment.

In this report a defect classification scheme is developed that is to support the control of the inspection of QUASAR requirements documents. In particular we discuss our approach to define the scheme, its underlying rationale, the scheme itself, and our experience from using it in a case study.

The scheme consist of two defect attributes that are to be used when inspecting use cases and recording defects in these documents, and three attributes for state chart documents.

Keywords: Defect, Defect Classification, Inspection, QUASAR

Table of Contents

1 Introduction

Software inspection plays a crucial role in achieving high quality software right from the beginning. Especially for requirements documents inspections are beneficial as defects can be detected and removed at an early point in time before they can leak into subsequent phases of the development process, where those defects can cause high rework cost and quality problems.

In order to fully exploit the beneficial aspect of inspections it is necessary to control them: Based on an analysis of the inspection effectiveness (e.g., what type or number of defects have been detected and what type and number might be remaining) it has to be decided whether development can proceed as planned or controlling actions, such as re-inspections, have to be performed.

Defect classification is a technique that can support this activity. Using appropriate defect classification schemes it is possible to get a quantitative statement of the inspection effectiveness (i.e., what type of defects have been found). Examples in the literature demonstrated beneficial applications but are hard to adopt without tailoring the underlying defect classification scheme to the environment.

In this report a defect classification scheme is developed that is to support the control of the inspection of QUASAR requirements documents. In particular we discuss our approach to define the scheme, its underlying rationale, the scheme itself, and our experience from using it in a case study.

2 Development of the Scheme

2.1 Objective of the Classification Scheme

There are various objectives that can be pursued in defining and applying a defect classification scheme: classified defect data can be used to characterize the development process, to improve it, or to control it [Freimut, 2002].

In this report we aim at developing a defect classification scheme that can be used to control software inspections in the requirements phase, specifically, the inspection of documents created using the QUASAR approach [Kamsties et al, 2001].

Pilot applications for controlling software inspections using defect classification data exist in the literature, namely the Orthogonal Defect Classification (ODC) approach developed at IBM [Chaar et al., 1993]. Using this approach the actual kinds of defects found are analyzed and compared with the expected kinds of defects.

By means of a specifically designed defect classification scheme the analysis allow to make inferences whether the type of defects is appropriate considering the current inspection or test activity. For example, during integration testing it is possible to decide whether the defects found are related to integration testing or whether they are related to unit testing. If the latter is the case, controlling actions in terms of additional unit testing can be launched.

For inspections the scheme was designed to relate the defects found to the skill level of inspectors. Thus it is possible to determine, whether, for example, major kind of defects have been not targeted by the inspectors and are therefore still undetected in the document.

A similar method is to be developed for the inspection of QUASAR documents starting with Use Cases and State Chart Diagrams [Kamsties et al, 2001].

2.2 Approach for the Development of a Scheme

In order to develop a defect classification scheme, generally two approaches are possible [Freimut, 2002].

Using a bottom up approach, real defects are investigated and analyzed in order to identify commonalities and differences and this insight is used to define attribute values for a defect classification attribute.

Using a top down approach, attribute values are derived based on theoretical properties and considerations.

Both approaches have their pro's and con's. Therefore, in order to harness the benefits of both approaches, we aim at a mixed strategy: In a first step we derive attributes and attribute values based on theoretical considerations. In a second step, actual defects are used to apply the defect classification scheme and revise the scheme as necessary.

Thus, the work is partially explorative in the sense that applying the devised scheme is expected to lead to new, additional insight into the applicability of the scheme that will influence the further development of it

The remainder of this chapter discusses our theoretical considerations resulting in the scheme presented in Chapter 3. Chapter 4 then discusses the empirical usage of the proposed scheme in a case study.

2.3 Rationale Underlying the Scheme

2.3.1 Design Rationales in Existing Work

The QUASAR defect classification scheme is to be used to support the control of software inspections. Since the Orthogonal Defect Classification (ODC) Scheme [Chillarege et al., 1992] is a defect classification scheme that specifically aims at controlling inspection and test activities, we use the principles underlying that scheme in order to develop the QUASAR scheme.

Within the ODC Scheme three attributes are analyzed in order to control inspection and test activities. These three attributes are listed in Table 1.

Attribute Name	Attribute Meaning	Attribute values
Trigger	How did you detect the defect?	Inspection Triggers[1]: Design Conformance, Logic/ Flow, Backward Compatibility, Lateral Compatibility, Concurrency, Internal Document, Language Dependency, Side Effect, Rare Situations
Defect Type	What had to be fixed?	assignment, checking, algorithm, function, timing, interface, relationship
Defect Qualifier		missing, incorrect, extraneous

Table 1 Attributes used to control inspections in IBM case study

The attribute Defect Type describes the language construct that had to be fixed when correcting the defect. The attribute Defect Qualifier refines the nature of

[1] The are also separate sets of attribute values for unit test and system test.

the correction by an indication, whether the defect was due to an omission, a commision, or extraneous. Finally, the attribute Trigger captures, what the inspector was thinking about when detecting the defect.

The attribute Defect Type is often used to control test activities [Freimut, 2002]. For each attribute value (cf. Table 1) exists an expectation when in the course of the development process a high number of defects related to that attribute value should be found and when a low number of defects should be found. Since such an expectation exists for all attribute values, there exists also an expectation for each inspection or test activity, how the distribution of attribute values in that activity should look like. If the actual attribute value distribution differs from this expected distribution, one can start exploring causes for such result and define controlling actions.

The attribute Trigger was applied in a case study project to control inspection activities [Chaar et al., 1993]. The attribute values describe certain ways how the document is scrutinized for defects. For each attribute value there exists a mapping, which skill is necessary to find defects related to that trigger value. If too few defects of a particular trigger value are detected, this is an indicator that certain skill levels where missing in the inspection. Depending on the amount of defects that are assumed to be not found, an re-inspection can be scheduled (i.e., the inspection process be controlled).

The general question with both defect attributes is, whether we can use the scheme or at least the underlying principles of the scheme in order to control the inspection of requirements documents of the QUASAR context as well.

There are two reasons why the existing scheme cannot be used directly for the QUASAR context. First, the attribute values for the attribute Defect Type have been defined for detailed design or code documents[2] but not for early requirements documents, which, however, are the focus of the QUASAR work. Second, the attribute values for the defect type Trigger reflect the way the inspectors read the document and check it for defects. Consequently, the attribute values are supposed to strongly depend on the reading technique used. Therefore, QUASAR, in which also reading techniques are investigated, requests also an adaptation here.

2.3.2 Design Rationales for the QUASAR Scheme

To design the scheme we decided that the QUASAR scheme is to have attributes that are equivalent to the three presented attributes of the ODC scheme. In the following these three attributes are discussed.

[2] Also for User documentation and natural language support.

Analogon to ODC Defect Type

The ODC attribute Defect Type has attribute values that describe what was fixed when correcting the defect. These attribute values consist mainly of structures that a code document consists of.

Therefore, the QUASAR scheme will contain an attribute Location, in which the various structures of a Use Case or a High-Level State Chart will be captured.

In a first version of the scheme, this attribute will contain the different aspects of a use case that refers to the use case entry that was corrected to fix the defects or the state chart item that was corrected. If experience shows that this is on too fine a level of granularity, coarser levels are possible.

The concrete refinement and definition of attribute values is performed in the next chapter.

Analogon to ODC Trigger

Within ODC the Trigger captures, what the inspector was thinking about when detecting the defect. This intention is very similar to the way the inspectors read the document and check it for defects. Consequently, the attribute values are supposed to strongly depend on the reading technique used.

At the time being, checklist-based reading (CBR) and perspective-based reading (PBR) are intended reading techniques for QUASAR documents.

For CBR the problem is, that inspectors can read the use cases in different ways, which makes it difficult without experience for the reading of use cases in inspections, to develop a set of attribute values. Moreover, when using PBR, the attribute values should be tailored to the reading approach, which on the other hand opens the question whether this will result in too specific a scheme.

Therefore, the rationale for a QUASAR attribute is to capture the quality aspect that is impacted by the defect. This rationale is a natural extension of the IBM rationale, as when thinking about defects, inspectors check for quality aspects.

This way both CBR and PBR can be supported as i) CBR the questions are defined according to quality criteria, ii) the questions in PBR scenarios are also used to check for quality properties explicitly defined in the scenario.

To adopt this strategy for the development of the QUASAR scheme it is necessary

1. to select and define quality aspects that a good requirements document should follow.
2. to define attribute values based on these definitions.

Regarding the first task, several listing of quality properties of requirements documents exist. For example, [Myers, 1979] defines a list of seven sins and the IEEE standard 830 lists a set of required quality aspects.

Since the sections of attribute values is always an important but also creative task, we report here our underlying rationales so that others may benefit from our thoughts: We used both these lists in order to select those quality aspects we deemed appropriate for our defect classification scheme. In selecting the quality characteristics we largely based our scheme on the IEEE830 list. An exception were the quality aspects changeability and traceablity. Our reason to do so was that in several instances it was hard to distinguish for a defect between these properties and other properties in the table. For example, if something is hard to understand, should it be a defect related to Understandability or to Changeability. Therefore, achieving orthogonal attribute values is one reason to omit changeability and traceability.

Another reason was that changeability and traceability can be addressed by means of the structure of a requirements document. The QUASAR document strucure is designed to keep information locally by putting required information in pre-defined sections of a requirements document. Additionally, traceability is also achieved by requiring certain information to be documented. Consequently, defects eventually impacting changeability or traceability are actually defects because required information according to the document structure, is not complete. Therefore, those defects can be classified as with the proposed set presented in Table 2.

Finally, we selected the quality aspects depicted in Table 2.

Quality Aspect	Definition
Correctness	Information presented in a requirements document is correct when it is equivalent to some reference standard. This reference is given by means of the expected or intended behavior of the system to be developed or documents developed earlier in the development process[3].
Completeness	Information presented in a requirements document is complete if no required pieces of information such as significant requirements, definitions of responses in all situations, and labels, figures, diagrams and definitions, are missing[4].
Testability	Information in a requirements document is testable if there exists an operational approach to check that the information represents the requirements.

[3] The IEEE830 standard refers to a correct requirements document, if every requirement stated is one that the software shall meet. Our definition is a more operational definition of that sense.

[4] The IEEE830 standard refers to a complete requirements document, if all significant requirements, all definitions of responses in all situations, and all labels, figures, diagrams and definitions are included.

Quality Aspect	Definition
Unambiguity	Information presented in a requirements document is unambiguous if every piece of information has only one interpretation.
Consistency	Information in a requirements document is consistent if there are no contradictions between pieces of information given in different places of the document.
Understandability	Information in a requirements document is understandable, if the reader can easily understand and comprehend the information in order to process it further.
Feasibility	Information in a requirements document is feasible if there exists a process that allows to transform the presented information into the form of executable software.
Redundancy/ Overspecification	Information presented in a requirements document is redundant or over specified if it is irrelevant or is too detailed in the sense that it antedates decisions that are to be taken later in the process.

Table 2 Quality Aspects of Good Requirements Documents

These general definitions for well-defined requirements are negated in order too derive definitions for attribute values for defects resulting from violating these quality properties. This is done in Table 3 on a high level. These general definitions are instantiated for use cases and high-level state charts in the next chapter.

Attribute Value	Description
Incorrectness	The behavior described in the document does not match to the expected or intended behavior. The information presented in the document is wrong.
Incompleteness	The behavior described does not contain all necessary scenarios. Information that is required for subsequent activities is not presented.
Untestability	The behavior described in the document can due to logical or physical constraints not be validated by means of a test case or by that fact that too many test cases would be needed. That means there is no operational way to check whether the system fulfills the requirements.
Ambiguity	The described behavior or presented information can be interpreted in two or more ways. Thus, it is not clear, which of the two or more interpretations are true.
Inconsistency	Aspects of the behavior or two or more pieces of information are described in at least two different, incompatible ways so that there is a contradiction between them.

Attribute Value	Description
Incomprehesibility	The presented information is difficult to understand and comprehend. Specific instances are also deviations from the prescribed document format.
Infeasibility	The described behavior cannot be implemented.
Redundancy/ Overspecification	Information is irrelevant or too detailed in the sense that it prescribes an implementation

Table 3 High-Level Definition

Analogon to the Defect Qualifier

Within ODC the Defect Qualifier refines the nature of the correction by an indication, whether the defect was due to an omission, a commision, or extraneous.

A similar structure is used for the QUASAR scheme. In addition to the three ODC attribute values, however, we add an attribute value 'alternative' as in requirements inspections often remarks are made requesting other solutions that are preferable to the described ones, which are nevertheless correct. This is similar to schemes in [Freimut et al., 2000] and the HP scheme [Grady, 1992].

Attribute Value	Description
Missing	The defect was due to an omission. Therefore, when correcting the defect, something had to be added. For example, an entire use case or state chart was missing.
Incorrect	The defect was due to a comission. Therefore, when correcting, something had to be changed. For example, a use case event flow does not capture the behavior the client wants.
Extraneous	The defect was due to something not relevant to the system. Therefore, it was deleted when correcting the defect. For example, the precondition was too stringent.
Alternative	The defect was due to something that had to be changed as there is a more beneficial way of doing it.

Table 4 Attribute Values for Defect Qualifier

Side Remark

The three attributes of the ODC scheme as well as our equivalents capture two dimensions: the location of a defect (given by Defect Type and Defect Qualifier) and the way how the defect was detected (given by Defect Trigger).

[Chernak, 1996] also regards these two dimensions as the most important ones for inspections: where do I have to look for defects and how can I spot it.

3 The QUASAR Defect Classification Scheme

In this chapter the proposed defect classification scheme is presented. Section 3.1 presents the attributes and their values for use cases. Section 3.2 presents the attributes and their values for state charts.

3.1 Defect Classification Scheme for Use Cases

3.1.1 Attribute Location

In structure of the SysRD document, which is used in QASAR as first requirements document, is depicted in Figure 1.

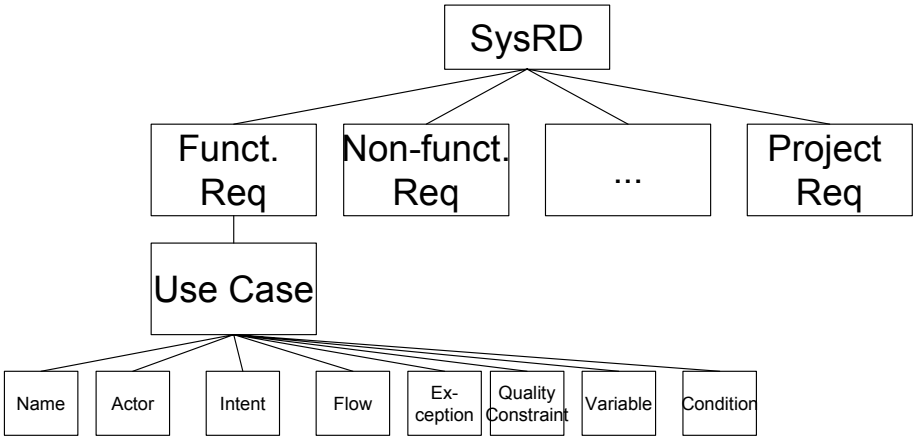

Figure 1 Structure of SysRD in order to identify elements usable as attribute values.

Based on this figure we derive the attribute values for the attribute Location as shown in

Attribute **Location** : What part of the use case did you fix in the first place when correcting the defect ?

Attribute value	Description
Name	The name of the use case was changed when correcting the defect.
Actor	The actor part of the use case was changed when correcting the defect.
Intent	The intent part of the use case was changed when correcting the

Attribute value	Description
	defect.
Flow	The flow part of the use case was changed when correcting the defect.
Exception	The exception part of the use case was changed when correcting the defect.
Quality Constraint	The quality constraint part of the use case was changed when correcting the defect.
Variable	The variable part (monitored or controlled) of the use case was changed when correcting the defect.
Condition	The condition part (pre, post) of the use case was changed when correcting the defect.
Case	An entire use case was changed (e.g., added, removed)
Other	Other locations. This encompasses for example the use case diagram.

Table 5 Attribute Value Definitions for QUASAR attribute Location in Use Cases

Comparing this list of attribute values to [Anda and Sjoberg, 2002], who also defined a defect classification scheme for use case models, reveals a large amount of commonality. They distinguish between Actors, use cases, event flow, variations, relationships between use cases and trigger/pre-/post-conditions.

3.1.2 Attribute Trigger

In this chapter, we tailor the general definition of the attribute trigger as described in Section 2.3.2 to use cases. Use cases are used in QUSAR to describe the functional requirements of a system in the SysRD.

Attribute Value	Description	Example
Incorrectness	The Use Case dose not match to the expected or intended behavior; that is, the information presented in the Use Case is wrong and does not represent the user requirements.	The flow of a Use Case does not represent the flow of activities expected by the user.
Incompleteness	The Use Case dose not contain all necessary scenarios. The Use Case set does not contain all necessary use cases. Information that is required for subsequent activities is not presented.	An important exception is not specified, a certain actor is not considered.

Attribute Value	Description	Example
Untestability	The behavior described in the Use Case cannot be validated by means of test cases due to logical or physical constraints. That means there is no way to check whether the system fulfills the Use Case.	It is impossible to derive the system response to a certain user input.
Ambiguity	Elements of the Use Case can be interpreted in two or more ways. Thus, it is not clear, which of the two or more interpretations are true.	The name of the monitored variable "environmental input" is ambiguous if there is more than one input from the environment.
Inconsistency	The information of a single Use Case or the information of different Use Cases is described in at least two different, incompatible ways so that there is a contradiction between them.	The quality constraints of a use case contradict the event flow. One user action in two different use cases results in contradictory system behavior.
Incomprehensibility (Not understandable)	The Use Case is difficult to understand and comprehend. The Use Case is not specified according to a given template.	The event flow described in the use case is too complex due to many include relationships. The template is not adhered to.
Infeasibility	The behavior described in the Use Case cannot be implemented.	It is not possible to derive high level statecharts from the use case.
Redundancy/ Overspecification	The Information given in the Use Case is irrelevant or too detailed in the sense that it prescribes an implementation	Details of the user interface are described in the use case. An actor not necessary for the system behavior is described in the use case.

3.2 Defect Classification Scheme for High Level State Charts

3.2.1 Attribute Location

In order to define attribute values for the attribute Location for High Level State Charts (HLSC), we derived the components from a High-Level State Chart document based on the guidelines presented in [Denger et al, 2002]. These components are shown in Figure 2.

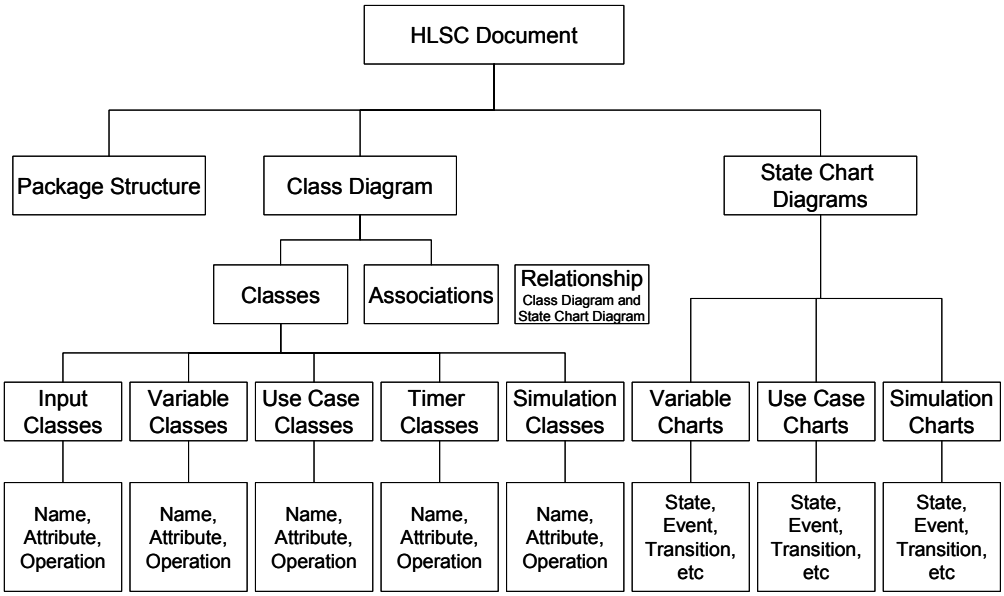

Figure 2 Structure of a HLSC Document

A HLSC document consists of a package structure, a class diagram (which might be omitted depending on the requirements tool used), and – mainly – state charts. Based on the guidelines in [Denger et al., 2002], classes are derived from use cases in the following way: actors become an (input) class, monitored and controlled variables become a (variable) class, the function-behavior of a use case becomes a (use case) class. Finally, classes for a timer and a simulation environment are identified. Classes from variables, use cases, and the simulation environment are refined with state chart diagrams. State chart diagrams and class diagrams, finally, consist of the typical structures of classes and state charts.

We found a structure overview as in Figure 2 useful to discuss the scope, contents, and granularity of a Location scheme.

This figure shows that various levels of granularity within the attribute values are possible. For example, it is possible to select the attributes on the second level of the picture (i.e., {Package Structure, Class Diagram, State Chart Diagram} or the lowest level, which would result in a large number of attribute values.

In order to select the granularity of the attribute values we considered the following context applicable in the QUASAR:

- The state chart diagrams contain most of the intellectual work of a HLSC document. Therefore, they should be represented with a higher degree of granularity in the scheme.

- The usage of class diagrams depends on the tool used for creating the state charts and is not mandatory for the QUASAR approach. The guidelines in [Denger et al, 2002] assumed a tool requiring class diagrams. However, in other cases class diagrams might be omitted. Therefore, the scheme should contain for completeness purposes also attribute values for class diagrams.

As a consequence, the attribute Location for statecharts actually consists of two attributes: Location and (for defects detected in state charts) State Chart Location. The attribute Location offers a general view of the location, whereas the State Chart Location goes more into detail of the state chart and therefore considers the fact that most of the intellectual work in included here.

Attribute Value	Definition
Package Structure	The defect was corrected by fixing the package structure
Class	The defect was corrected by fixing a class description or a class in a class diagram
Association	The defect was corrected by fixing an association
Relationship	The defect was correcting by fixing the relationship between a class description or diagram and the state diagrams
Variable State Chart	The defect was corrected by fixing a state chart implementing a monitored or controlled variable.
Use Case Chart	The defect was corrected by fixing a state chart implementing a use case.
Simulation Chart	The defect was corrected by fixing a state chart implementing the simulation environment.

Table 6 Attribute Value Definitions for Attribute Location (HLSC Documents)

These attribute values aim at capturing some semantic aspects of the system by relating to the kind of state chart diagram that was fixed, but by including also the class information.

The refinement of the state chart defects is seen in the following table.

Attribute Value	Definition
State	The defect was corrected by fixing a state in a state chart diagram.
Event	The defect was corrected by fixing an event in a state chart diagram.
Transition	The defect was corrected by fixing a transition in a state chart diagram.
other	All other defects

Table 7 Attribute Value definition for State Chart Location (HLSC document)

Note that this attribute is only filled in, if Location is of type Variable Chart, Use Case Chart, or Simulation Chart.

3.2.2 Attribute Trigger

In this chapter, we tailor the general definition of the attribute trigger as described in Section 2.3.2 to High Level Statecharts (HLSC). HLSCs are used in QUSAR to describe the functional requirements of a system in the SysSD.

Attribute Value	Description	Example
Incorrectness	A statechart diagram or a set of diagrams is does not represent to the Use Cases in the SysRD.	The states of the statechart do not represent the states mentioned in the use case.
Incompleteness	A statechart diagram or a set of diagrams does not contain all necessary use case elements. Information that is required for other activities is not presented.	The statechart model does not represent all user requirements; essential states, events, actions, operations, and guard conditions are missing.
Untestability	A statechart diagram or a set of diagrams is untestable, if there exists no operational process to check that the statecharts fulfill their requirements. That means there is no way to check whether the statechart fulfills the requirements[5].	The expected system reaction in response to a certain event cannot be derived from the statechart model.
Ambiguity	Elements of the statecharts can be interpreted in two or more ways. Thus, it is not clear, which of the two or more interpretations are true.	Ambiguous event names: The event name "environmental input". In the case of more than one input from the environment, it is impossible to decide which event is meant in a certain situation.
Inconsistency	The information of a single statechart diagram or the information of different statecharts is described in at least two different, incompatible ways so that there is a contradiction.	The actions triggered by the same event, in two different statecharts, result in contradictory system behavior.

[5] Within the QUASAR context statecharts are executable and therefore per definition testable. However, we include this attribute value for completeness reasons.

Attribute Value	Description	Example
Incomprehensibility (Not understandable)	The statechart diagrams are difficult to understand and comprehend. The statecharts are not specified according to a given template.	The statecharts are too complex due to many relationships between the statecharts.
Infeasibility	The behavior described in the statecharts cannot be implemented.	It is not possible to derive low level statecharts from the high level statecharts.
Redundancy/Oversp ecficication	The Information given in a statechart diagram or a set of statecharts is irrelevant or too detailed in the sense that it prescribes an implementation.	Details of the user interface are described in the statecharts. A state not necessary for the system is described in a statechart diagram.

4 Evaluation of the Defect classification scheme

The defect classification scheme described in this report has been evaluated as a part of a case study that analyzed the performance of different requirements inspection techniques. In this chapter the lessons learned and the perceived applicability of the defect classification scheme are described.

4.1 The Objective of the Case Study

In industrial settings many success stories can be found about the effectiveness and the efficiency of software inspections. In order to optimize the inspection approach different reading techniques such as checklist-based reading and scenario-based reading have been proposed. Various experiments have been performed to evaluate which of these techniques produces better inspection results; that is finds more defects with less effort. Scenario-based reading approaches performed better then ad-hoc or checklist-based approaches in some experiments but failed to improve the checklist-based approaches in others. Thus, the success factors of scenario based reading approaches need to be further analyzed. In order to empirically evaluate the impact of the detailed descriptions provided by scenario-based approaches and the active involvement of the inspectors on the inspector's efficiency and effectiveness we conducted a case study to compare checklist-based reading and perspective-based reading with respect to the influence of the detailed reading scenarios. In order to evaluate the applicability of the defect classification scheme the scheme was used during the inspection performed in the case study to classify the defects found in the requirements documents.

4.2 The Context of the Case Study

The case study was part of the practical course "Software Engineering 1" at the university of Kaiserslautern. During this course the subjects had to develop a building automation system that regulates the temperature and the light in rooms and floors of a university building. In the working description of the practical course the system was separated into three sub-systems, namely, the "Temperature Control" (Temp), "Light Control" (Ligh) and "User Interface" (UI) system.

The students had to develop the whole system following the V-Model, starting from a general problem description of the system and ending with the acceptance testing of the complete system. Within this development process the students had to develop use cases and scenarios based on a problem description

of the system. After the creation of the use cases and the scenarios the students performed inspections of the created documents. 12 subjects were involved in the case study and a group of 4 people was responsible for one subsystem. Each group inspected the two sub systems of the other groups. As part of the inspection process the students had to document the detected defects and each defect had to be classified with the defect classification scheme presented in this report.

One part of the case study was a debriefing questionnaire the students had to respond to after the use case inspections were completed. One part of this questionnaire aimed to evaluate the perceived applicability of the defect classification scheme. The results of the evaluation of the questionnaire are presented in the following section. For further information regarding the context of the case study and the empirical approach refer to [Ciolkowski et al., 2003]

4.3 Results of the Evaluation

The subjects had to answer three questions regarding the perceived applicability. These questions focus on the ease of use of the classification scheme and whether it is possible to uniquely assign a defect location and a defect class to a detected defect. In addition the subjects had to specify aspect of the defect classification that are not clear to them.

In Figure 3 the subjects perception of the ease of use is presented. One important aspect of a defect classification scheme is that it can be easily used during the inspection process to classify the detected defects. If the defect classification is cumbersome or difficult to apply the inspectors would refuse to use it in their work or even worse would randomly choose a defect classification.

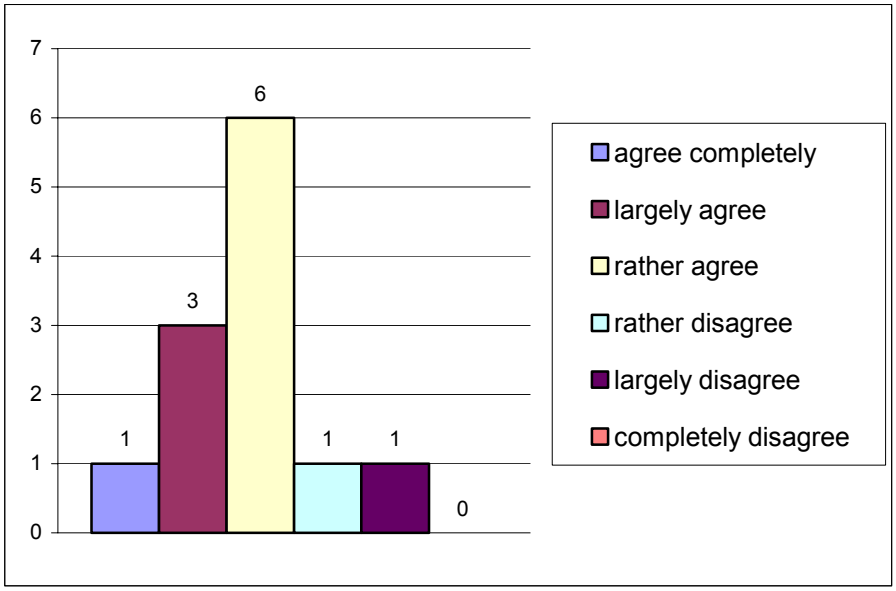

Figure 3: Is the usage of defect classification simple?

The evaluation shows that most of the subjects (10 out of 12) perceive the defect classification as easy to apply. Only 2 subjects stated that they perceive the defect classification as rather difficult to use. This indicates that the defect classification scheme can be easily used during a use case inspection to classify the detected defect.

A second crucial aspect of a defect classification is that it is possible to assign unique values to the different attributes of the classification. Thus, the subjects had to estimate whether it was possible to assign a unique defect location and a unique defect class to the different defects they identified in the use case inspections.

In Figure 4 the results for the defect location are presented. Again most of the subjects (9 out of 12) state that it is possible to assign a unique defect location to a detected defect.

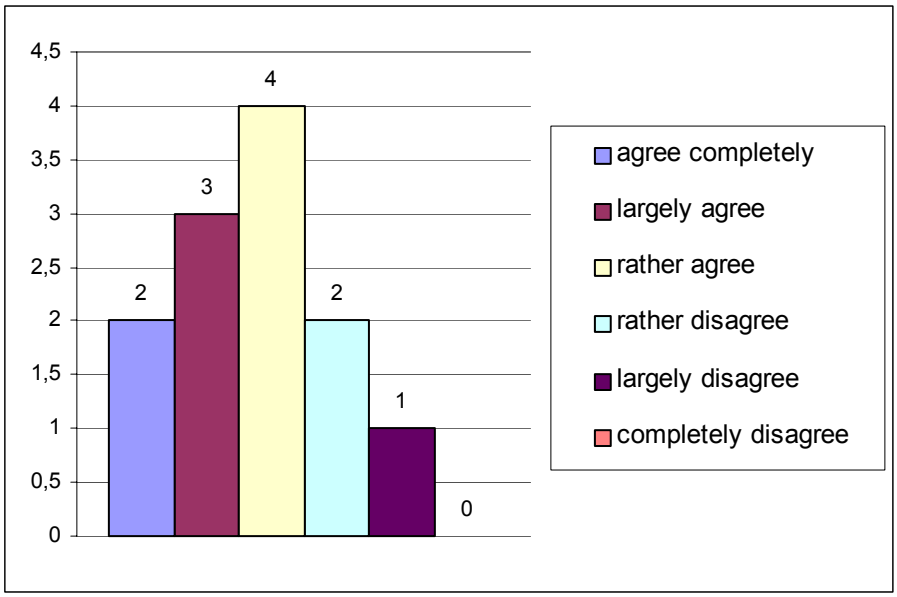

Figure 4: Is it easy to assign a unique defect location to a defect

Also most of the subjects state that is possible to assign a unique defect class to a detected defect (7 the students out of 12). However, regarding this attribute of the classification the result of the evaluation is not that conclusive as for the defect location attribute.

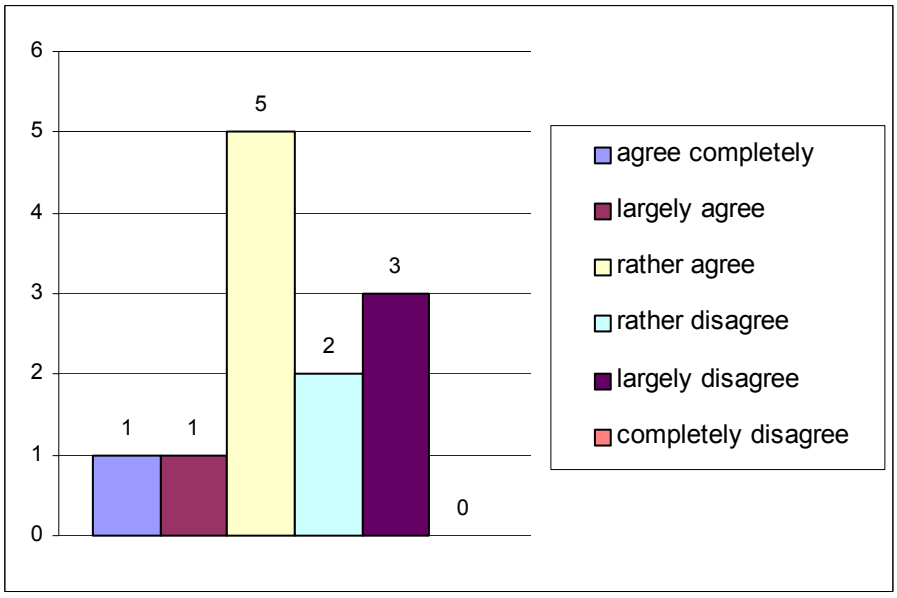

Figure 5: Is it possible to assign a defect to a defect-class explicitly?

Taking the evaluation of both classification attributes into account the case study indicates that it is possible to uniquely classify a defect with the scheme.

In order to support the subject's perception of the classification scheme they had to state disadvantages of the scheme, which got aware when using it in the inspections. Unfortunately only a few subjects responded to this question and most of the answers are not valuable to improve the defect classification scheme. One basic result is that the subject's perceive the classification scheme incomplete with respect to the given defect classes. Three subjects stated that they were not able to identify a class for a defect; that is, they miss certain attribute values for the defect class attribute. The subjects did not mention which defects could not be assigned to a certain class and a detailed analysis of the defect report form also does not indicate which defects cannot be assigned as each defect was classified in the scheme.

Another lesson learned is that the subjects had problems with the definition of the attribute value "ambiguity" and "infeasibility". Both definitions were not absolutely clear as the subjects stated that ambiguity fits to every defect and infeasibility is not applicable as in the early phase of the project it is too difficult to make decisions about the feasibility of a requirement. This critique is supported by an expert's evaluation of the defect report form. This evaluation indicates that some defect classes were not suitable for the corresponding defect they are assigned to.

Furthermore, this detailed analysis of the defects shows that the most frequently used defect classes are incompleteness, inconsistency and incorrectness. Ambiguity, redundancy and incomprehensible are sometimes used while infeasibility and untestability were never used. The reason for this might be the poor quality of the created use cases with respect to completeness and correctness issues. However, the definitions of the attribute values untestability and infeasibility will be checked for improvement possibilities.

To summarize this brief evaluation of the defect classification there is an indication that the classification scheme is applicable in a practical context of use case inspections. The majority of the subjects perceive the scheme as easy to use and also perceive it possible to assign unique attribute values to detected defects.

Of course the results of this case study are neither significant nor generalizable but they give a first impression of weaknesses and advantages of the classification scheme. In further research activities the definitions of the attributes of the defect classification should be improved to enable a unique classification of the defects. Moreover, the classification scheme should be used in a bigger case study or experiment to evaluate its applicability.

5 References

Bente Anda and Dag Sjoberg, Towards and Inspection Technique for Use Case Models, SEKE 2002, pp. 127-134, 2002.

Inderpal Bhandari, Michael J. Halliday, Jarir Chaar, Ram Chillarege, Kevin Jones, Janette S. Atkinson, Clotilde Lepori-Costello, Pamela Y. Jasper, Eric D. Tarver, Cecilla C. Lewis, and Masato Yonezawa, In-process improvement through defect data interpretation, IBM Systems Journal, vol. 33, no. 1, pp. 182--214, 1994.

Barry Boehm, Vic Basili, Defect Reduction Top 10, IEEE Software, January, 2001.

Lionel C. Briand, Bernd Freimut, Oliver Laitenberger, Guenther Ruhe, and Brigitte Klein, Quality Assurance Technologies for the EURO Conversion - Industrial Experience at Allianz Life Assurance, in Proceedings of the 2nd International Software Quality Week Europe, 1998.

Jarir K. Chaar, Michael J. Halliday, Inderpal S. Bhandari, and Ram Chillarege, On the Evalutation of Software Inspections and Tests, in Proceedings International Test Conference, pp. 180--189, 1993.

Yuri Chernak, A Statistical Approach to the Inspection Checklist Formal Synthesis and Improvement, IEEE Transactions on Software Engineering, vol. 22, pp. 866--874, Dec. 1996.

Ram Chillarege, Inderpal S. Bhandari, Jarir K. Chaar, Michael J. Halliday, Diane S. Moebus, Bonnie K. Ray, and Man-Yuen Wong, Orthogonal defect classification -- A concept for in-process measurements, IEEE Transactions on Software Engineering, vol. 18, pp. 943--956, Nov. 1992.

Jarir K. Chaar, Michael J. Halliday, Inderpal S. Bhandari, and Ram Chillarege, In-Process Evaluation for Software Inspection and Test, IEEE Transactions on Software Engineering, vol. 19, pp. 1055--1070, Nov. 1993.

Markus Ciolkowski; Christian Denger; Filippo Lanubile. Empirical Evaluation of the Success Factors of Perspective Based Reading. Technical Report IESE 2003 to be published.

Christian Denger, Daniel Kerkow, Antje von Knethen, Maricel Medina Mora, Barbara Paech, Richtlinien: Von Use Cases zu Statecharts in sieben Schritten (in German), Technical Report, Fraunhofer IESE, 2002.

Norman E. Fenton and Shari Lawrence Pfleeger, Software Metrics - A Practical and Rigorous Approach. International Thomson Computer Press, 2nd edition ed., 1996.

Bernd Freimut, Brigitte Klein, Oliver Laitenberger, and Günther Ruhe, Experience Package from the ESSI Process Improvement Experiment HYPER, Tech. Rep. IESE-Report 015.00/E, Fraunhofer Institut für Experimentelles Software Engineering, 2000.

Bernd Freimut, Developing and Using Defect Classification Schemes, Technical Report, Fraunhofer IESE, 2002

Robert B. Grady, Practical Software Metrics for Project Management and Process Improvement. Prentice Hall, 1992.

Institute of Electrical and Electronics Engineers, IEEE Recommended Practice of Requirements Engineering, IEEE Std. 830-1993, 1993.

E.Kamsties, A. von Knethen, B. Paech: Structure of QUASAR Requirements Documents, IESE-Report No. 073.01/E, 2001

G. Myers, The Art of Software Testing, John Wiley & Sons, 1979.

Document Information

Title:	Developing and Using Defect Classification Schemes
Date:	September 1, 2003
Report:	IESE-076.03/E
Status:	Final
Distribution:	Public