Fraunhofer Institut
Experimentelles
Software Engineering

Developing and Using Defect Classification Schemes

Author:
Bernd Freimut

Funded by the German BMBF
under grant VFG0004A ("QUASAR")

IESE-Report No. 072.01/E
Version 1.0
September 1, 2001

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

Defects play a crucial role in software development. This is because on one hand, defects, when detected, should be corrected so that the final version of the developed software artifact is of higher quality. On the other hand, defects carry a lot of information that can be analyzed in order to characterize the quality of processes and products, to track the progress of a project and control it, and to improve the process.

Therefore, defect measurement plays a crucial role in many software measurement programs. Consequently, in many measurement programs defect data are collected. Generally there are several pieces of information that can be collected about defects. The most often used pieces of information relate to the quantity of defects (i.e., their number) and their type. For the latter one, defect classification schemes are used to quickly characterize the nature of defects.

Two important questions arise, when using defect classification: "How can a defect classification be designed?" and "How can defect classification data be analyzed?"

In order to answer the first question, this report presents the aspects of a defect that have been measured in the literature and it presents the possible structures of a defect classification scheme. Finally, examples of frequently used defect classification schemes are presented.

In order to answer the second question, this report presents general methods to analyze defect classification as reported in the literature as well as concrete analyses for a variety of purposes.

This report is a result of the project "Integrierte Qualitätssicherung und Anforderungsanalyse zur Softwareentwicklung im Umfeld Fahrzeug (QUASAR)", which is funded by the German BMBF under the grant VFG0004A.

Keywords: Defect, Defect Classification, Inspection, Testing, Measurement Program

Table of Contents

1 Introduction

Defects play a crucial role in software development. This is because on one hand, defects, when detected, should be corrected so that the final version of the developed software artifact is of higher quality. On the other hand, defects carry a lot of information that can be analyzed in order to characterize the quality of processes and products, to track the progress of a project and control it, and to improve the process.

Therefore, defect measurement plays a crucial role in many software measurement programs. Consequently, in many measurement programs defect data are collected. Generally there are several pieces of information that can be collected about defects. The most often used pieces of information relate to the quantity of defects (i.e., their number) and their type. For the latter one, defect classification schemes are used to quickly characterize the nature of defects.

Defect classification schemes are used by organizations of low and high maturity. For example, [Paulk et al., 2000] report that many high maturity organizations (CMM level 4 and 5) use defect classification (esp. orthogonal defect classification) for their quantitative management. But also when implementing measurement programs in companies with smaller CMM-levels, defect classification is used very frequently (e.g., [Briand et al.,1998])

Two important questions arise, when using defect classification. The first question is how a “good” defect classification scheme can be defined. This question is of practical relevance as many companies want to define and use their own classification scheme, which is specifically tailored to their needs and goals. However, in practice such self-defined classification schemes often impose problems due to ambiguous or overlapping attribute types, or by capturing different aspects of a defect in the same attribute, as also reported by [Ostrand and Weyuker, 1984], [Hirsh et al., 1999]. Thus, a systematic approach to define defect classification schemes would be useful.

The second question is, for what purposes defect classification data can be analyzed and how this analysis can be performed. This question is on one hand interesting because in the set-up of a measurement program the potential for measurement goals answerable with defect classification should be known. On the other hand, knowing methods to analyze defect classification data allows to fully exploit the information and address the measurement goals.

To address these two questions, this report presents the results of a literature survey on defect classification schemes.

The basis of this survey were papers identified by means of a query in the literature database INSPEC with the keywords “defect” and “classification”. Moreover the internet was searched with various search engines using the keywords “defect classification” and “software”.

The literature search, however, proved difficult as only few papers specifically deal with defect classification schemes. Often defect classification schemes are used as a tool for investigating other aspects (e.g., in the domain of software inspections). These reports could therefore not be systematically included in this report.

In contrast to existing surveys on that topic ([Fredericks and Basili, 1998], [Kuhröber, 1997]), which focus more on an in-depth discussion of single defect classification schemes, this survey aims at providing a more aggregate discussion on the definition and usage of defect classification schemes.

2 Terminology

This section introduces the terminology used throughout this report.

A *failure* is a departure of the system behavior from its required behavior. Failures can be observed when the system is executed such as in testing or during field usage.

A *fault* is uncovered when either a failure of the program occurs or an internal error (e.g., an incorrect state) is detected within the program. The cause of the failure or internal error is said to be a fault.

An *error* is a human action resulting in software with a fault.

The term *defect* is used in a generic manner referring to either a fault or a failure, provided the distinction is non-critical.

An *Attribute* is a feature or property of a defect that we are interested in [Fenton and Pfleeger, 1996]. Using defect classification, this attribute is measured on a nominal or ordinal scale with a set of pre-defined values. These values are called *Attribute Values*.

3 Defect Classification Schemes

3.1 Scope of classification: what can be classified

There are many aspects of a defect that might be relevant for analysis. Defects are inserted due to a particular reason into a particular piece of software at a particular point in time. The defects are detected at a specific time and occasion by noting some sort of symptom and they are corrected in specific way. Each of these aspects (and more) might be relevant for a specific measurement and analysis purpose.

[Mellor, 1992] and [Fenton and Pfleeger, 1996] proposed a framework[1] of defect key elements that capture on a high-level different aspects of a defect. These key elements of a defect have been chosen to be (as far as possible) mutually independent (i.e., orthogonal).

Each of the framework's key elements can be refined leading to many attributes of a defect that can be captured by means of measurement in the form of defect classification. In order to give the reader an idea on what might be possible to measure, explain the key elements and illustrate them with concrete attributes from of existing, common defect classification schemes[2].

Location

The location of a defect describes where in the documentation the defect was detected. This information can be very detailed and capture a within-system identifier (e.g., document name or module identifier). However, the attribute can also contain attribute values describing different high-level entities of the entire system (e.g., Specification/Requirements, Design, Code, Documentation, etc.) or describe the faulty document in more detail such as its age or history.

Timing

The timing of a defects refers to phases when the defect was created, detected, and corrected.

An attribute like *Detection Phase* can capture the phase during which the defect was detected. Another aspect of timing beside the detection of a defect is the question of when the defect was created and first introduced in the system. This information is usually captured in an attribute *Origin* that contains process phases as attribute values.

[1] The framework can be instantiated differently for faults and failures. In this context we instantiate it in terms of faults.

[2] In several instances the mapping was hard to perform (e.g., for symptom and end result). Therefore, the mapping of attributes to key elements is considered to be subjective.

Symptom

Symptom captures what was observed when the defect surfaced or the activity revealing the defect. For example, the ODC attribute *Trigger* captures the mechanism that allows a defect to surface. For instance, during inspections the inspector classifies a defect according to what he was thinking when detecting the defect while a tester classifies according to the purpose of the test case revealing the defect.

Under symptom it is also possible to classify what is observed during diagnosis or inspection. For example, the attribute *Type* of the IEEE classification scheme provides a very detailed classification of the symptom.

End result

End result describes the failure caused by the fault. For example, the ODC attribute *Impact* captures the impact of a fault (or resulting failure) on the customer. This attribute contains values such as Performance, Usability, Installability, etc.

Mechanism

Mechanism describes how the defect was created, detected, and corrected[3]. *Creation* captures the activity that inserted the defect into the system. The *Activity* captured the activity that was performed when the defect was detected (e.g., inspection, unit test, system test, operation). Finally, the correction refers to the steps taken to remove the defect. For example, the ODC attribute *Type* is explicitly defined in terms of activities performed when correcting defects.

Many defect classification schemes contain attributes that describe the creation or correction of defects in terms of omission and commission. (e.g., schemes for inspections, HP scheme and attribute mode, and ODC scheme defect qualifier.) These can also be seen as describing how a defect was created and corrected.

Cause

Cause describes the error leading to a fault. For example [Mays et al., 1990] uses attribute values like Education, Oversight, Communication, Tools, and Transcription for an attribute *Cause*. [Leszak et al., 2000] uses different attributes capturing different kind of causes: *Human-Related Causes* (e.g., lack of knowledge, communication problems, etc), *Project Causes* such as time pressure or management mistake. Finally, *Review Causes* describe why the defect slipped potentially through an inspection (e.g., no or incomplete inspection, inadequate participation, etc)[4].

Severity

Severity describes the severity of a resulting or potential failure. For example, it might capture whether a fault can actually be evidenced as a failure.

---

3 In contrast to the key element Timing, which captures process phases, the element mechanism captures activities.

4 In addition to the presented three attributes, Leszak et al. use the attribute *Phase Causes* to capture the phase in which the defect was introduced (thus, being equivalent to the Origin attribute described earlier).

Cost

Cost capture the time or effort to locate/isolate a fault and correct it. Typically, such information is captured not by means of a classification but by means of a ratio-scale. However, it is also possible to capture time or effort data on an ordinal scale (thus, being able to use classification).

Table 1 summarizes this information by mapping attributes of the schemes presented in Section 3.2 to the key elements. The attributes are organized according to contain the defect classification schemes in which the attribute is defined (ODC, IEEE, HP, other).

Framework Key Element	Attributes of existing schemes			
	ODC	HP	IEEE	other
Location of a defect describes where in the system the defect was detected	Target, Age, History	Origin	Source	
Timing of a defects refers to phases when the defect was created, detected, and corrected			Project Phase	Origin[5],
Symptom captures what was observed when the defect surfaced or the activity revealed the defect	Trigger	Type	Type, Product Status, Symptom	
End result describes the failure caused by the fault	Impact		Repeatability, Impact Attributes, Product Status	
Mechanism describes the how the defect was created, detected, and corrected	Type, Activity, Defect Qualifier	Mode	Project Activity, Corrective Action	
Cause describes the error leading to a fault			Suspected Cause , Actual Cause	Cause[6], Causes[7]
Severity describes the severity of a resulting or potential failure				
Cost capture the time or effort to locate/isolate a fault and correct it				

Table 1 Mapping of framework's key elements to attributes of existing defect classification schemes

3.2 Existing Schemes

In this section examples of common defect classification systems are presented. The purpose of the presentation is to give the reader who want to re-use existing or develop new defect classification schemes appropriate references and to introduce appropriate examples for the subsequent sections.

The rationale of selecting the schemes was to take common schemes that have been often used and reported. In particular, we present one defect classifica-

5 [Freimut et al., 1998]

6 [Mays et al., 1990]

7 [Leszak et al., 2000]

tion scheme proposed by an IEEE standard and two industrial schemes for which several types of analyses have been published and that have been referenced in general software engineering books (e.g. [Pfleeger, 1998]). Additionally, two defect classification schemes specifically for inspections are presented.

In particular, we present the IEEE Standard Classification for Software Anomalies [IEEE,1994], the Hewlett-Packard Scheme [Grady, 1992], and the Orthogonal Defect Classification Scheme [Chillarege, 1992],[IBM].

3.2.1 IEEE Standard Classification for Software Anomalies

The IEEE scheme is aimed for audiences who want to implement a defect classification scheme compliant to a standard or who want to expand a defect tracking or defect classification scheme and are looking for proven methodologies supporting that effort [IEEE,1994].

The different attributes of the scheme are organized according to a general defect classification process consisting of four steps.

The first step *Recognition* occurs when the defect is found. In the second step, *Investigation*, the defect is investigated in sufficient depth either to identify all unknown issues and propose solutions or indicate that the defect requires no action. In the third step, *Action*, a plan of action is established to resolve the defect and prevent it from occurring. The last step, *Disposition*, is performed when all required resolution actions are completed or at least identification of long-term corrective actions have been completed.

Table 2 shows the attributes of the entire scheme. The table contains the steps of the defect classification process, the attribute names as well as their meaning. Finally it is indicated, whether the attributes are mandatory (m) or optional (o) in order to be IEEE standard compliant.

Defect Process	Attribute Name	Attribute Meaning	mandatory
Recognition	Project Activity	What were you doing when the defect occurred?	m
	Project Phase	In which life-cycle phase is the product?	m
	Suspected Cause	What du you think might be the cause?	o
	Repeatability	Could you make the defect appear more then once?	o
	Symptom	How did the defect manifest itself?	m
	Product Status	What is the usability of the product with no changes?	o
Investigation	Actual Cause	What caused the anomaly to occur?	m
	Source	Where (part of the system and its documentation) was the origin of the defect?	m

Defect Process	Attribute Name	Attribute Meaning	mandatory
	Type	What type of defect/enhancement at the code level?	m
Action	Resolution	What to do to prevent the defect from happening again?	m
	Corrective Action	What action to take to resolve the defect?	m
Impact Identification	Severity	How bad was the defect in more objective engineering terms?	m
	Priority	Rank the importance of resolving the defect (taking subjectively into account all other Impact attributes)?	o
	Customer Value	How important is a fix to the customer?	o
	Mission Safety	How bad was the defect wrt. Project objectives or human well-being?	o
	Project Schedule	Relative effect on the project schedule to fix?	m
	Project Cost	Relative effect on the project budget to fix?	m
	Project Risk	Risk associated with implementing a fix?	o
	Project Quality/ Reliability	Impact to the product quality or reliability to make a fix?	o
	Societal	Impact of society of implementing the fix?	o
Dispostion	Disposition	What actually happened to close the anomaly?	m

Table 2 Attributes of IEEE scheme

The attribute values for the attributes are not shown here but can be referred to in [IEEE,1994]

3.2.2 Hewlett-Packard Scheme

The HP scheme was developed by HP's Software Metrics Council in 1986 [Grady, 1992]. The purpose of the entire scheme is to improve the development process by reducing the number of defects over time [Pfleeger, 1998].

The developers use this scheme by selecting three descriptors for each defect: the Origin (i.e., where was the defect injected in the system), the type of defect, and the mode (i.e., whether information was missing, unclear, wrong, changed, or done in a better way).

The entire scheme with attributes and attribute values is shown in Figure 1. The bold lines in this picture denote the possible selection of attribute values. As it will be described in Section 3.3, the choice of an attribute value for the attribute *Origin* defines the possible set of attributes available (and reasonable) for the attribute *Type*.

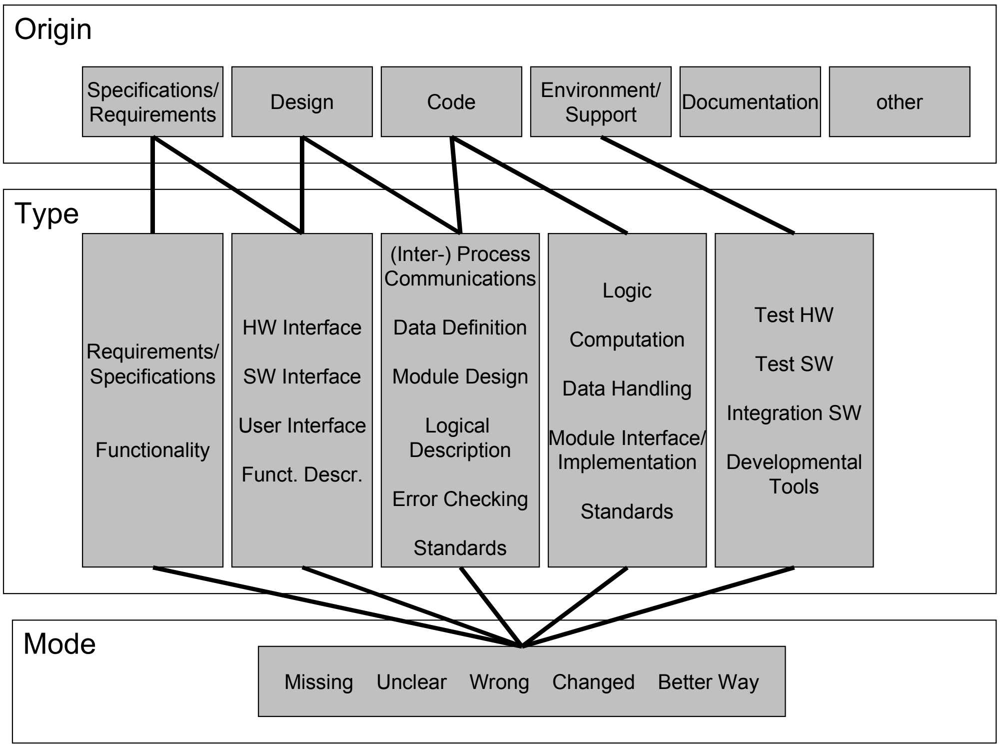

Figure 1 HP Defect Classification Scheme.

3.2.3 Orthogonal Defect Classification

Orthogonal Defect Classification has been developed by IBM [Chillarege, 1992]. This defect classification scheme can also be used to improve the entire development process by reducing the number of defects over time [Pfleeger, 1998]. However, the original, and most often reported purpose is to give project teams feedback on the progress of the current project.

Since its definition in [Chillarege, 1992] this classification scheme has been adopted by more and more organizations, as shown by an increasing number of experience reports ([Sreenivasan, 1999], [Silberman, 1998], [Schultz, 1999], [Hirsh et. al, 1999], [Dalal et. al., 1999], [Bridge and Miller, 1997], [Amezquita and Siewiorek, 1996], [Rentschler, 1995], [Humphrey, 1995]). In the survey performed by [Paulk et al., 2000], 14 out of 37 high-maturity organizations (i.e., being of CMM level 4 or 5) used this scheme as quantitative analysis practice.

Table 3 shows the attributes of the ODC scheme for design and code defects, their meaning, and the corresponding attribute values. Extensions of the ODC scheme exist for information development, graphic user interface, the build process, and national language support. These extensions introduce additional sets of attribute values for Defect Type and Trigger that are available depending on the attribute value of the attribute Target. These extensions, however, are not discussed here.

The attributes are organized according to the two process steps, in which the defect classification data are collected. The process step Open is performed when a defect has been detected and a new defect report is opened in the defect tracking system. The process step close is performed when the defect has been corrected and the defect report is closed.

Process	Attribute Name	Attribute Meaning	Attribute values
Open	Activity	When did you detect the defect?	design inspection, code inspection, unit test, integration test, system test
	Trigger	How did you detect the defect?	Design Conformance, Logic/ Flow, Backward Compatibility, Lateral Compatibility, Concurrency, Internal Document, Language Dependency, Side Effect, Rare Situations
			Simple Path, Complex Path, Coverage, Variation, Sequencing, Interaction
			Workload/Stress, Recovery/Exception, Startup/Restart, Hardware Configuration, Software Configuration, Blocked Test
	Impact	What would have customer noticed if defect had escaped into the field?	Installability, Serviceability, Standards, Integrity/Security, Migration, Reliability, Performance, Documentation, Requirements, Maintenance, Usability, Accessibility, Capability
Close	Target	What high level entity was fixed?	requirements, design, code[8], build/package/merge, information, user-interface
	Source	Who developed the target?	in-house, library, out-sourced, ported
	Age	What is the history of the target?	base, new, rewritten, re-fixed
	Defect Type	What had to be fixed?	assignment, checking, algorithm, function, timing, interface, relationship
	Defect Qualifier		missing, incorrect, extraneous

Table 3 ODC Scheme (for Design and Code)

While all these attributes capture the semantics of a defect [Chillarege, 1992] and are useful to analyze, the attributes defect type and trigger play a crucial role in the scheme.

[8] For targets Natural Language, Build/Package/Merge a set of own defect types exist. For target Interface an own set of triggers exist

The attribute *Defect Type* captures the fix that was made to resolve the defect. One interesting aspect was taken into account when developing the set of attribute values: For each defect type an expectation exists, in which detection activity (e.g., unit test, function test, system test) that defect type should be detected. If an analysis of the defects reveals that the activities are not finding the right types of defects, obviously, these processes need to be improved [Kelsey, 1997] and within the project it has to be reacted upon this result.

For example, defects of type Function are those that require a formal design change. It can be expected that the number of Function-defects decreases over time. This profile of an attribute value over time (i.e., different detection activities) is called a signature.

Based on such a signature, deviations from the expected trend can be identified. For example, if the number of Function-defects is increasing with testing time, then a problem might exist that should be investigated.

Since each attribute value has a signature of its own, each detection activity has a specific distribution of attribute values that is expected. The progress of the project can be measured against these expected distributions, as shown in Figure 2.

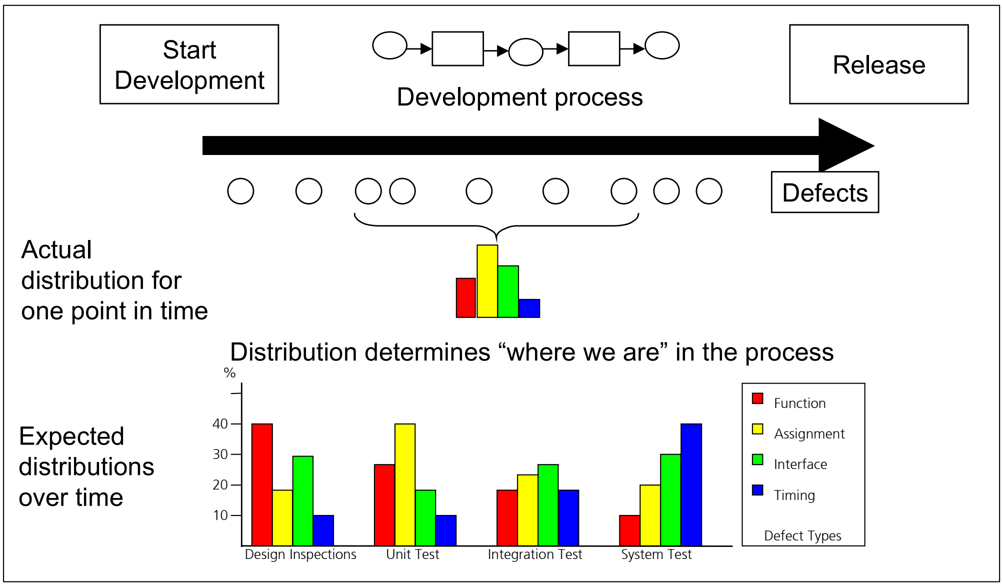

Figure 2 Signatures of ODC Attribute Defect Type

The second attribute special to ODC is the attribute Trigger. This attribute captures the reason why a particular piece of software was executed when the fault caused a failure. This illustrated in Figure 3.

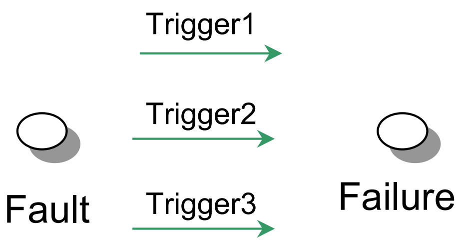

Figure 3 Concept of trigger.

At first, a defect is dormant in the system. Then the code is activated (e.g., automatically via test cases or mentally in inspections) and the fault turns into a failure. The facilitator activating dormant software faults into failures is called the trigger and there can be several facilitators why the code is executed (physically or mentally).

In practice, the data collector assigns an attribute value according to the question "What were you thinking about when you detected the defect?" (for inspections) and "What was the purpose of the test case?" (for testing).

Thus, the attribute trigger gives feedback to the defect detection process as it captures how defects are revealed.

3.2.4 Defect Classification for Inspections

While the schemes presented above aim to be used for several life-cycle phases, there are also defect classification schemes specifically developed for investigating inspections. These classifications can be more detailed than schemes supporting several life cycle phases as they can be tailored to the type of document being inspected and to the environment.

To give the reader a flavor of schemes used, we present one scheme for requirements documents and one for design documents.

As an example for a defect classification scheme for requirements we present the scheme proposed in [Porter et al., 1995]

1st level name	2nd level name	Definition
Omission	Missing Functionality	Information describing the desired internal operational behavior of the system has been omitted.
	Missing Performance	Information describing the desired performance specification has either been omitted or described in a way that is unacceptable for acceptance testing.

1st level name	2nd level name	Definition
	Missing Environment	Information describing the required hardware, software, database, or personnel environment in which the system will run has been omitted.
	Missing Interface	Information describing how the proposed system will interface and communicate with objects outside the scope of the system has been omitted.
Commission	Ambiguous Information	An important term, phrase or sentence essential to the understanding of the system has either been left undefined or defined in a way that can cause confusion and misunderstanding.
	Inconsistent Information	Two sentences directly contradict each other or express actions that cannot both be correct or cannot both be carried out.
	Incorrect or Extra Functionality	Some sentence asserts a fact that cannot be true under the specified conditions.
	Wrong Section	Essential information is misplaced within the document.

For design inspections little information is currently available in terms of defect classification. [Shull et al., 1999] proposed a scheme, by adapting a scheme proposed by [Baslili et al, 1996] for requirements for design documents.

Attribute Value	Definition
Omission	One or more design diagrams that should contain some concept from the general requirements or from the requirements document do not contain a representation for that concept.
Incorrect Fact	A design diagram contains a misrepresentation of a concept described in the general requirements or the requirements document.
Inconsistency	A representation of a concept in one design diagram disagrees with a representation of the same concept in either the same or another diagram.
Ambiguity	A representation of a concept in the design is unclear, and could cause a user of the document (developer, low-level designer, etc.) to misinterpret or misunderstand the meaning of the concept.
Extraneous Information	The design contains information that, while perhaps true, does not apply to this domain and should not be included in the design.

When comparing the attribute values for design inspections that have been proposed to classify defects with the attributes proposed by other defect classification schemes it becomes obvious that the attribute values are similar to the concepts captured from the ODC attribute Defect Qualifier.

Those defect classification schemes specifically developed for code inspections often either capture an ODC-Defect Type-like information, or capture the structure of the code that contained the defect, similar to the ODC Defect Type and the IEEE defect type

3.3 Structure of defect classification schemes

Measurement via classification is performed by assigning a measurement variable (i.e., an attribute) a discrete value, which is selected based from a pre-defined set of values (i.e., attribute values).

Defect classification schemes can differ in the way different attributes or attribute values relate to each other.

One early approach to structure defect classification schemes was to place a given defect in an appropriate node in a tree of categories. The primary characteristic of such a tree scheme is the requirement to place each error in a unique category that simultaneously represent all its features [Ostrand and Weyuker, 1984]. However, the many features of a defect and the resulting large number of categories make such schemes difficult to use.

While tree-based classification schemes aim at organizing different attribute in a tree-structure, it is also possible to organize the attribute values of one attribute in a hierarchical way. Thus, the attribute values on one level of the hierarchy are refined by the attribute values on the next level of the hierarchy. An example of this structure is for example given by the scheme in [IEEE, 1994] as shown in an excerpt in Table 4.

Attribute	Attribute Value		
Type	Logic		
		Forgotten cases or steps	
		Duplicate logic	
		Extreme Conditions neglected	
		…	
	Computation Problem		
		Equation insufficient or incorrect	
			Missing computation
			Operand in equation incorrect
			…
		Precision Loss	
			Rounding or truncation fault
			Mixed modes
		Sign convention fault	
	…		

Table 4 Example of hierarchically organized attribute values [IEEE, 1994]

Finally, a derivate of tree-based classification schemes are those defect classification schemes where the attributes are not independent but the selection of attribute values in one attribute influences the selection of attributes in a second attribute.

An example of such a scheme is the scheme used at Hewlett-Packard [Grady, 1992] as shown in Figure 4

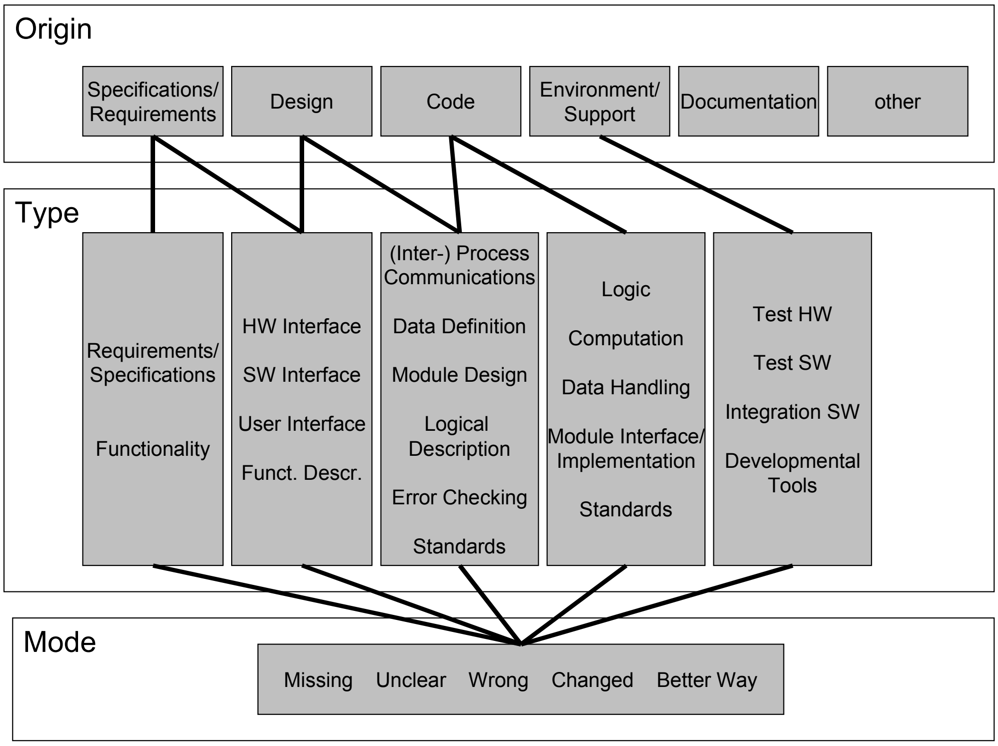

Figure 4 Scheme in which attributes are not independent

Based on the selection of an attribute value for the attribute *Origin*, different sets of attribute values for the attribute *Type* are possible.

A somewhat different approach is an attribute categorization scheme [Ostrand and Weyuker, 1994]. In such a scheme the various attributes whose values describe a defect are similar to the attributes of a relation in a relational database. Thus, the attribute values are assigned independently for each attribute. The ODC scheme for design and code [Chillarege et al., 1992] or the IEEE scheme are examples of such a scheme as shown in Figure 5.

Impact		Defect Type		Qualifier		Target	
Installability		Assignment/ Initialization		Missing	X	Code	X
Integrity/ Security		Checking		Incorrect		GUI	
Performance	X	Algorithm/ Method	X	Extraneous			
Maintenance		Function/ Class/ Object					
Serviceability		Timing/ Serialization					
Migration		Interface/ O-O Messages					
Documentation		Relationship					
Usability							
Standards							
Reliability							
Requirements							
Accessibility							
Capability							

Figure 5 Attribute categorization scheme (ODC excerpt)

The advantage of such a scheme is that new attributes (i.e., aspects of a defect) can be easily added.

3.4 Properties of defect classification schemes

As reported in [Ostrand and Weyuker, 1984], defect classification scheme often have problems including incomplete, ambiguous, and overlapping attribute types, too many attribute types, and a mixture of causes, symptoms, and actual faults for one attribute. Such problems make it difficult to correctly and reliably classify a specific defect resulting in poor quality data, which also question the results obtained from defect data analysis.

To prevent such problems, a defect classification scheme should possess several quality features as described below. These features should allow for an easy classification process, which is the prerequisite for high-quality data and analyses.

Orthogonal attributes and Orthogonal attribute values

The set of attribute values for a specific attribute should be orthogonal. This means that for a particular defect only one attribute value is appropriate.
If the attribute values are not orthogonal, it may happen that two or more attribute values may fit so that the data collector has arbitrarily to decide which value to assign. This, of course, leads to inconsistent and unreliable data.

An example of such problems is shown below according to [Fenton and Pfleeger, 1996]. Often organizations try to provide a single attribute for defects rather than using several attributes as depicted in Section 3.1. Consider the following classification for defects based on severity:

{major, minor, negligible, documentation, unknown}

This classification is not orthogonal as, for example, a documentation problem could also be a major problem. Consequently, there is more than one attribute value appropriate. The reason stems from mixing two different aspects of the defect, in this case Severity and Location.

To prevent this, orthogonal attributes should be developed. As described in Section 3.1, a defect has many aspects. Orthogonal attributes ensure that different aspects of a defect are captured in different attributes.
If one attribute contains two or more different aspects of a defect, the attribute values for that attribute can not be orthogonal. The problem most often occurring in practice is that cause, type, and impact are mixed. Thus, for a defect several attribute values (e.g., one for the cause, one for the impact) can be appropriate.

Complete attribute values

The set of attribute values should be complete so that for all defects an appropriate attribute value can be selected.
If the set of values is not complete, data collectors may decide not to classify the defect or select "the nearest" possible value resulting in inconsistent or incomplete data.
This problem can be easily prevented by using an attribute value "Other", for which the data collector has to textually describe the attribute value missing attribute values. This information can then be used to check whether the set of attribute values is actually incomplete and how to modify it.

Small number of attribute values

The number of attribute values should not be too large. If the number of attribute values is small, there is a greater chance that the human mind can accurately resolve between them. Having a small set to choose from makes the classification process easier and less error prone [Chillarege, 1992].
The number of attribute values depends on the purpose of the scheme. If the scheme is to be used by developers in their daily work, classification should be able to be performed easily and quickly. Consequently, only few attribute values are possible. If, on the other hand, the scheme is used by researchers, who have usually more time for a more careful analysis, a larger number of attribute values are possible.

Description of attribute values

The attribute values should be well-defined by means of a textual description. Often it can be seen in practice that the "definition" consists only of the title of the attribute value (e.g., logic). Therefore, it is easily possible to confuse or differently interpret different attribute values, which results in inconsistent and unreliable data. Therefore, each attribute value should be well-defined and augmented with example defects [Freimut et al, 2000]. An example of such definition is given in the ODC Standard [IBM] as shown below.

> Attribute Value: Checking
> Errors caused by missing or incorrect validation of parameters or data in conditional statements. It might be expected that a consequence of checking for a value would require additional code such as a do while loop or branch. If the missing or incorrect check is the critical error, checking would still be the type chosen.
>
> Examples:
> 1) Value greater than 100 is not valid, but the check to make sure that the value was less than 100 was missing.
> 2) The conditional loop should have stopped on the ninth iteration. But it kept looping while the counter was <= 10.

All these properties of a classification scheme (i.e., orthogonal, complete, well-defined, not too many attribute values for orthogonal attributes) are to make the classification process easy and therefore, improve the reliability of the data. Under reliability of a defect classification system it is understood that different people assign the same attribute value to the same defect. [El Emam and Wieczorek, 1999] propose a procedure to systematically and quantitatively assess the reliability of defect classification schemes.

In addition to these necessary properties, there are additional properties. For example, [Chillarege et al., 1992] asks for consistent defect classification schemes across process phases and products.

Consistency across phases means that the same classification scheme is used throughout all development phases. This feature eases classification as people have only to memorize one scheme for a project or product. If different schemes are applied for different phases, people might confuse the schemes used, especially when process stages of subsequent releases overlap. Moreover, consistency across phases allows also to look at trends across stages.
Consistency across products and projects means that the same scheme is used in different projects. This also eases classification for developers working in several projects since they only need to memorize one scheme. Moreover, it is then possible to analyze the data in order to identify process-independent defect patterns and obtain useful relationships and models [Chillarege et al., 1992].

4 How to develop Defect Classification Schemes

4.1 Strategies in Developing Defect Classification Schemes

If an organization wants to introduce a new classification scheme there are generally two strategies. The first one is to employ an existing defect classification scheme and tailor it to the organization's context. The second one is to develop a new scheme based on the objectives and needs of the organization.

Re-use of an existing scheme

Employing an existing classification schemes is particularly interesting for companies who start with defect measurement and simply want to characterize the defects found or do not want to invest effort in the development of a company-tailored scheme.

Schemes often adopted are the IEEE standard classification scheme [IEEE. 1994] and the Orthogonal Defect Classification scheme [Chillarege, 1992]. Especially for the latter more and more experience reports from organizations adopting ODC are being published ([Sreenivasan, 1999], [Silberman, 1998], [Schultz, 1999], [Hirsh et. al, 1999], [Dalal et. al., 1999], [Bridge and Miller, 1997], [Rentschler, 1995], [Humphrey, 1995].

Also the work of [Florac, 1992] aims at providing a re-usable defect classification scheme. In this report a principal set of measurable, orthogonal attributes are described. Checklists are provided supporting the construction of defect classification schemes by simply checking off attributes and values that are to be included for a specific context.

However, when re-using existing defect classification schemes they have nevertheless to be tailored to the organization. For example, the IEEE standard classification scheme explicitly assumes that for all attribute values organization-specific definitions have to be provided, a fact that is sometimes overlooked in practice. And also the ODC scheme (attribute defect type, trigger) employed, for example, at Tandem [Rentschler, 1995] has been modified against the original IBM scheme [Chillarege, 1992]. In addition, [Silberman, 1998] provides a very detailed study on transferring ODC into a different context.

Define new scheme based on measurement goals

In order to define a new classification scheme, rather than re-using and adapting existing schemes, the aspect to be captured in an attribute has to be selected and appropriate attribute values have to be defined.

The attributes to be included and their appropriate values are to be selected so that useful analyses are possible and the underlying measurement goals are achieved.

Typically, to select the attribute to be measured the framework developed by [Mellor, 1992] and presented in Section 3.1 can give guidance or apply "who, what, why, when, where, and how" questions [Fenton, 1991].

In order to define the attribute values generally two approaches are possible. Either based on *supposition* or based on *empirical evidence*.

Using supposition the attribute values are designed following pre-defined rationales. For example, in the design of the attribute defect type in the ODC scheme the attribute values were selected according to the activities performed when correcting a defect. Using empirical evidence the descriptions of actual defects are used to define the attribute values. For example, in [Ostrand and Weyuker, 1984] data collection captured textual descriptions of defects and their symptoms and used these descriptions as a basis for the definition of the classification scheme.

A helpful strategy to define attribute values is to use a rationale for selecting the attributes and their values. First, it should be clear what aspect of the defect the attribute is to capture. Once this aspect has been clearly defined, it will also be less likely that different aspects are mixed into one attribute violating the orthogonality of the scheme.

For example, in the ODC scheme, [Chillarege et al., 1992] the attribute defect type is to describe how to *fix* a (design and code) defect. The attribute values were then selected according to the *activities* the developers typically perform when fixing a defect.

[Chillarege et al, 1992] state that such a classification that is performed according to the activities the developers perform, ease the task of assigning attribute values (as the assignment is based on that task just performed) and therefore increase the quality and reliability of the defect classification data.

However, once an initial design for a defect classification scheme exists, modifications based on experiences have to be expected [Chaar et al, 1993].

4.2 A Process for Re-Using and Introducing Defect Classification Schemes

A process for the re-use of a defect classification scheme is presented in [IEEE, 1994]. This process is intended for organizations that want to implement the IEEE Standard Classification for Defect Anomalies on their projects or that want to expand an existing defect tracking system and are looking for proven methodologies supporting that effort.

Although this process is described in the context of re-using existing defect classification schemes, it can also be used as a basis for developing an entirely new scheme.

We briefly summarize this process in the following.

1. Select the attributes to be considered based on an intended standard compliance, based on the potential usage of the data, and based on the value for the organization's business.
2. For each attribute, determine the list of attribute values to be used. In this step first, attribute values are selected and an actual set of defects is classified according to them to check whether the attribute values make sense. If necessary, new attribute values have to be defined for special cases not considered in the proposed set of attribute values.
3. Document the attributes. Describe what is meant by each attribute using the organization's terminology and references.
4. Document the attribute values. Write sentences or paragraphs that exactly describe what the attribute values mean.
5. For each attribute value, determine, who is going to collect it and when.
6. Determine how the data can be collected by means of an existing defect tracking system.
7. Plan the kind of analyses you want to perform and when.
8. Provide training on the classification scheme to users and management.

5 How to analyze defect classification data

Already during the development of a defect classification scheme, the required and potential analyses are to be taken into account. For this purpose this section describes how defect classification data can be analyzed. The focus is on examples that have been reported in the literature.

5.1 General methods to analyze defect classification data

Charts

The most common form to analyze defect data is to determine their absolute and relative frequency. To visualize this distribution data often pie charts, histograms, or Pareto charts are used, for example as shown in Figure 6.

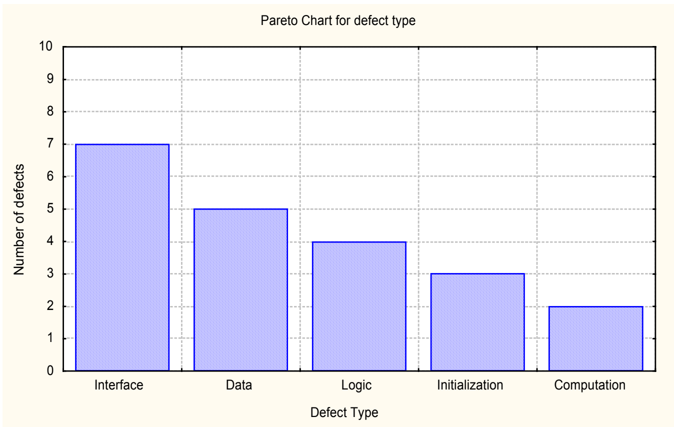

Figure 6 Pareto Chart to analyze defect data.

The appeal of such graphs is that they can be easily understood by developers [Silberman, 1998].

Comparison with Hypotheses

In order to control defect detection activities (i.e., inspections, testing), the distributions can be compared to expected distributions [Chillarege et al.,1992], [Chaar et al, 1993], [Bassin et al, 1998]. These expected distributions either stem from qualitative expectations on what attribute values should have a low or high number of occurrences or they can stem from historical baselines.

Data Mining

While it is relatively straightforward to compare the actual distribution of one attribute with its expected distribution, this is more difficult if several attribute or even cross-products of two or more attributes are to be analyzed. To support this kind of analyses [Bhandari et al., 1994], [Bhandari, 1993] proposed an automatic procedure based on data exploration techniques. This so-called Attribute Focusing Method aims at comparing automatically the actual relative frequencies of attribute values or attribute value pairs with expected relative frequencies and selecting those deviations that seem to be most interesting.

An illustration, taken from [Bandhari et al., 1994], is shown in Table 5 and Table 6.

Attribute Qualifier	Observed Frequency (%)	Expected Distribution (%)	Difference (%)
Incorrect	37	50	-13
Missing	63	50	13

Table 5 Illustration of Attribute Focusing (one attribute)

In Table 5 attribute focusing with one attribute is illustrated. For each attribute value the actual relative frequency is determined. Next, the expected relative frequency is computed by assuming a uniform distribution of all attribute values of a given attribute. Then the difference between the actual and expected distribution is computed. Finally, all attribute values are listed in decreasing order of the differences. A so-called filtering function selects then from top the attribute values (and hence the attributes) to be displayed in a chart. This filtering is performed in order to keep the number of analyses manageable.

Attribute Defect Type	Observed Frequency Defect Type (%)	Attribute Origin	Observed Frequency Origin (%)	Observed Frequency Type AND Origin (%)	Expected Frequency Type AND Origin (%)[9]	Difference (%)
Document	14	CLD	16	5	2	3
Document	14	MLD	84	8	11	-3
Assignment	16	MLD	84	16	14	3
Assignment	16	CLD	16	0	3	-3

Table 6 Illustration of Attribute Focusing (Two Attributes)

In Table 6 the computations are shown for two attributes. Again, for each attribute value the actual relative frequency is determined. Then, also for the attribute value pairs the relative frequency is determined. Next, the expected relative frequency of the attribute value pair is computed assuming statistical independence between the attribute values. Thus, the expected frequency of the attribute value pair is computed by multiplying the relative frequencies of

[9] Computes as "Observed Frequency Defect Type" x "Observed Frequency Trigger"

each attribute value. Similarly for the case of one attribute, the differences are then computed. Finally, all attribute value pairs are listed in decreasing order of the differences and can be selected by the above-mentioned filtering function.

Statistical Tests

Another way of analyzing defect classification data is to perform statistical analyses as proposed in [IEEE, 1994]. In this standard it is shown, for example, how to apply chi-square tests in order to test statistically, whether the attribute values of a given attribute are distributed uniformly. A second application is to test statistically, whether two attributes are statistically independent. However, in the literature considered in this survey none of these statistical analyses have been performed.

5.2 Interpretation of Analyses

One very important aspect of a defect data analysis is the interpretation of the analysis results by the data providers and the project team in the context of which the data were collected. While the methods described above merely to interesting situations in the project or process, only people with sufficient background and context information can relate the analysis results to causes of the detected patterns.

Two explicit processes of interpreting analysis results have been reported by [Bhandari, Halliday, et al., 1994] and [Matthews, 1999].

The interpretation process of [Bhandari, Halliday, et al., 1994] consists of the following steps:

<u>Result</u>: What is the observation made by means of the analyses?
<u>Cause</u>: What situations in the project might have yielded the observed results?'
<u>Implication</u>: What is going to happen, if no action is initiated?
<u>Action:</u> What is to be done to prevent negative consequences?
<u>Validation:</u> The action has been performed, check whether it produced the expected results.

The interpretation process of [Matthews, 1999] looks similar:

<u>Result:</u> Describe each unusual result briefly in words. Collect the facts from several results together and make a summary in plain, understandable words.
<u>Cause</u>: Elaborate on the summary to identify the story that explains the results. Try to confirm the story or decide alternative interpretations. Talk to key people involved in the project.
<u>Action</u>: When the story seems solid, propose specific actions to address any issues.

5.3 Specific Analyses

While the previous section discussed several techniques to analyze defect classification data in general, this section presents actual measurement goals that have been achieved by analyzing defect classification data.

The purpose of the section is on one hand to give the reader an idea what possible measurement goals can be achieved using defect classification. On the other hand, when the reader has a concrete measurement goal in mind, the corresponding section presents attributes that can be defined and also analyses that can be performed.

The measurement goals presented in the following are not intended to be orthogonal. Rather, they are organized according to the intention of the source describing the analysis.

5.3.1 Characterization of the defects found

The simplest purpose of analyzing defect classification data is to characterize the defects found. Characterization in this context means that the number and proportion for each attribute value is known.

This type of analysis goal is often employed by companies starting measurement program who want to obtain a first overview of the defects. For example, in [Briand et al., 1998] the defects found in newly introduced inspections were classified according to the ODC-impact in order to characterize the defect found in terms of their visibility to the user.

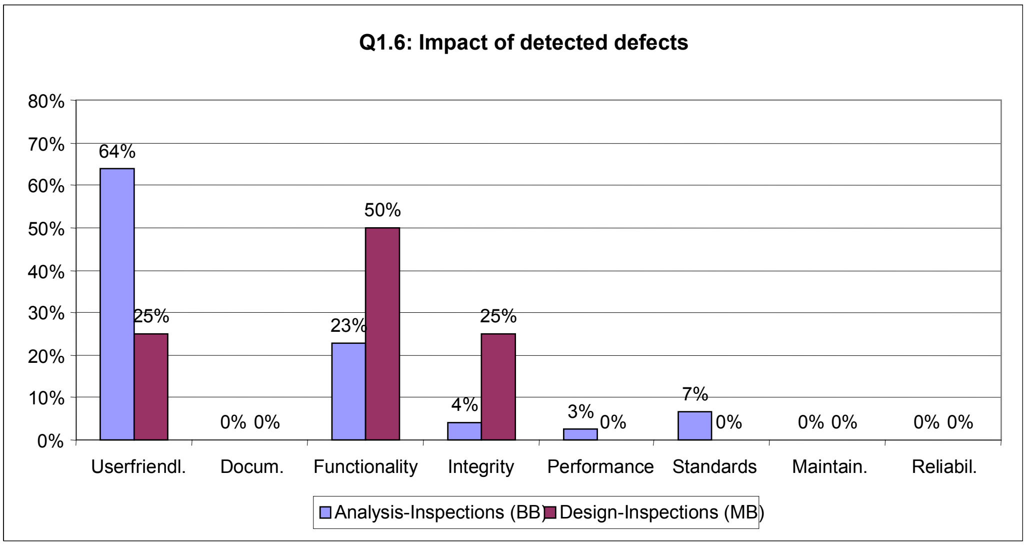

Figure 7 Simple characterization of defects with bar chart

Typically, some sort of improvement (see Section 5.3.2) or some sort of evaluation of the employed technologies [Briand et al., 1998] can then be initiated based upon the results.

5.3.2 Defect Prevention

One important purpose of analyzing defect data is to learn what kind of defects are made and to use this knowledge to prevent similar defects in the future. The technique for analyzing the data is called Defect Causal Analysis [Card, 1998] or Defect Prevention Process [Mays et al., 1990].

The idea of Defect Causal Analysis (DCA) is to analyze defect data in order to find a systematic error. In this context, a systematic error is an error that is committed repeatedly and therefore causes many faults and failures. Once the systematic error is identified, its underlying cause can be identified and appropriate process changes can be performed to prevent the systematic error and the resulting faults and failures. Overall, this techniques aims to prevent defects or at least enable an earlier detection.

One important part of this technique is the identification of the systematic error. For this purpose, the analysis of defect classification data is used. Based on a Pareto-Chart, the type(s) of defects occurring most often are identified. These types of defects typically hint to the systematic error.

Depending on the actual implementation of DCA in an organization either defect reports of a sample of the corresponding defects are analyzed in a qualitative manner [Card, 1998] to identify the systematic error or directly the identified types are used for reasoning about the systematic error [Grady, 1992].
According to [Chillarege et al., 1992], [Matthews, 1999] the advantage of the latter approach is, that it is supposed to take less time. The disadvantage of a qualitative analysis is that few defects have to be discussed in detail in a team of developers, whereas an analysis of defect classification data can take into account all defects and provide results in short time.

In the following, an example from [Grady, 1992] is shown for illustrating purposes. The defect data were collected during maintenance in a Hewlett-Packard division developing systems software (using the language C, SA/SD design, informal and sporadic code inspections, branch coverage testing, regression testing).

The defects were classified according to the HP scheme presented in Section 4. Figure 8 shows the defect classification data.

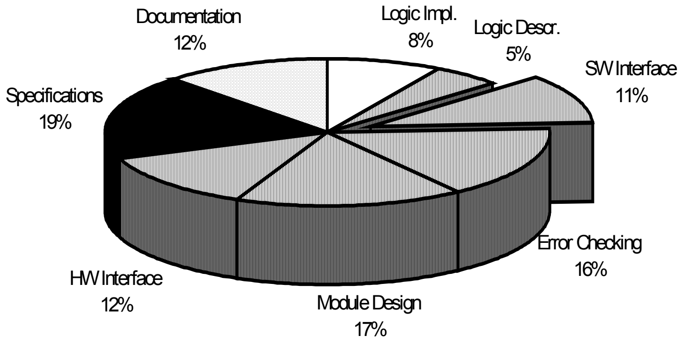

Figure 8 Defect data (type what)

This picture shows that the majority of defects, as indicated by the striped wedges, have their origin in the design phase. Therefore, one potential systematic error could be in that phase. However, [Grady, 1992] not only takes into account the number or proportion of an attribute value to make a decision about systemic errors, but also the value obtained by normalizing with the average effort to find and fix a defect.

To do so, for all defects of a given attribute value the effort to find and fix the defects is determined. This overall effort is then divided by the number of defects with the given attribute value to obtain the average effort to find and fix a defect of the given attribute value. The cost of the given attribute value is then the number of defects with that attribute value multiplied by the average effort to find and fix those defects.

In Figure 9 such data is shown for the example.

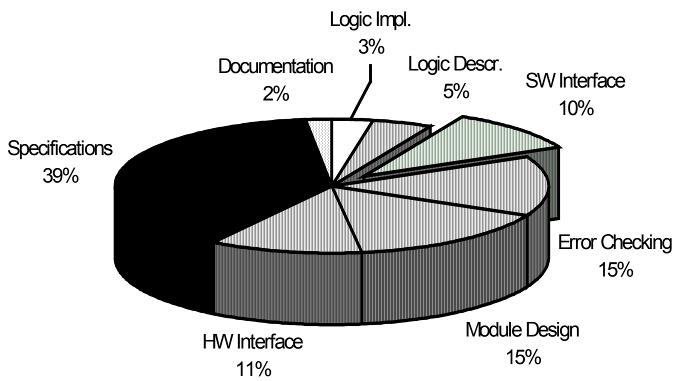

Figure 9 Normalized defect data

This figure shows that specification defects make a large proportion and therefore offer much improvement potential, especially for new products. Consequently, it was decided to prevent specification-defects in the future.

Since from the type of defects (i.e., specification) the systematic error was not obvious, a brainstorming was performed in the course of which a Fishbone-Diagram, as shown in Figure 10, was developed.

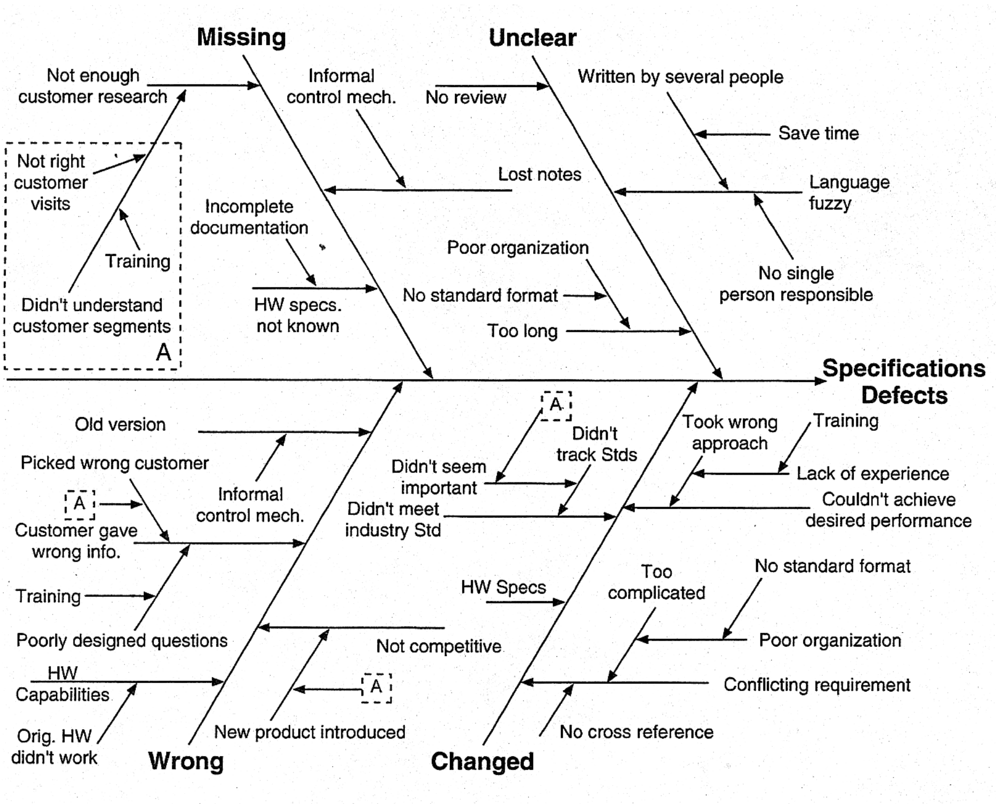

Figure 10 Fishbone-Diagram to identify root cause

As root causes from Figure 10 were identified: an unclear understanding of customer segments, a lack of clearly assigned responsibilities, the absence of document standards, and several changes in the hardware.

To improve the process (and prevent future defects) it was decided to have the marketing department to set up customer visits in order to learn about customer needs, to assign the configuration management responsibility to one person who is to select a tool for version control, and to set up task force for developing a new specification documentation format.

This example is narrated in a straightforward manner. In practice, however, typically several attributes are investigated in an explorative manner to investigate a meaningful subset of data.

Often, the crucial attribute(s) used in the course of a DCA is the attribute or attributes capturing the cause of a defect. Such a classification (e.g., [Mays et al., 1990] or [Leszak et al., 2000] aims at more directly pinpointing to the root causes.

5.3.3 Control Inspections

Defect classification can be used to assess the effectiveness of software inspections and use the gained knowledge to control inspections. The approach followed in [Chaar et al., 1993] is to analyze the actual kinds of defects found and compare them with the expected kinds of defects.

An important attribute for this type of analysis is the ODC attribute trigger. For inspections this attribute captures the way the defect was detected (i.e., the classification in performed based on the question: What were you thinking about when you detected the defect?).

An example of this type of analysis is given in [Chaar et al., 1993] and shown in Figure 11. This figure shows data obtained from a high-level design inspection of a middleware component.

The attribute shown is the ODC-trigger. Two particular attribute values for this attribute are *Lateral Compatibility* and *Backward Compatibility*. A defect is assigned the value Backward Compatibility when the inspector uses extensive product/component experience to identify an incompatibility between the function described by the design document or the code, and that of earlier versions of the same product or component. From a field perspective, the customer's application, which ran successfully on the prior release, fails on the current release. A defect is assigned the value Lateral Compatibility when the inspector with broad-based experience, detects an incompatibility between the function described by the design document or the code, and the other systems, products, services, components, or modules with which it must interface.
It is clear from the description of the attribute values that different skills are necessary to detect defects with both triggers. To detect Lateral Compatibility defects, the inspector needs broad-based experience with the systems the inspected system has to interface with. On the other hand, to detect Backward Compatibility defects, the inspector needs experience within the inspected system across several releases.

As it can be seen in Figure 11, only a small proportion of defects was found with the Lateral Compatibility trigger. For the data analysts this was surprising as they expected beforehand many more defects of Lateral Compatibility as the inspected system was a middleware component, which, naturally, has to interface with other systems.

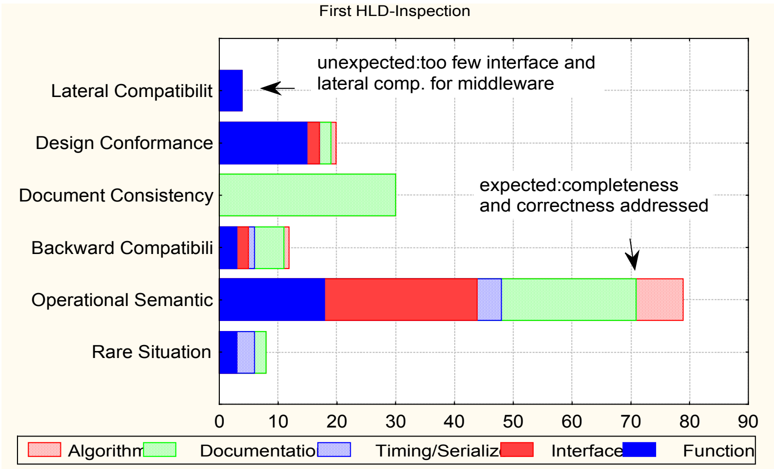

Figure 11 Data for ODC-trigger from High-Level-Design Inspections of middleware component

The interpretation of this deviation given by the project team was that mainly inexperienced inspectors inspected the document. Therefore, there was potential that many more defects addressing compatibility issues were still undetected.

Consequently the project team decided to re-inspect the system using experts who specifically concentrated on compatibility issues. The result of the re-inspection is shown in Figure 12.

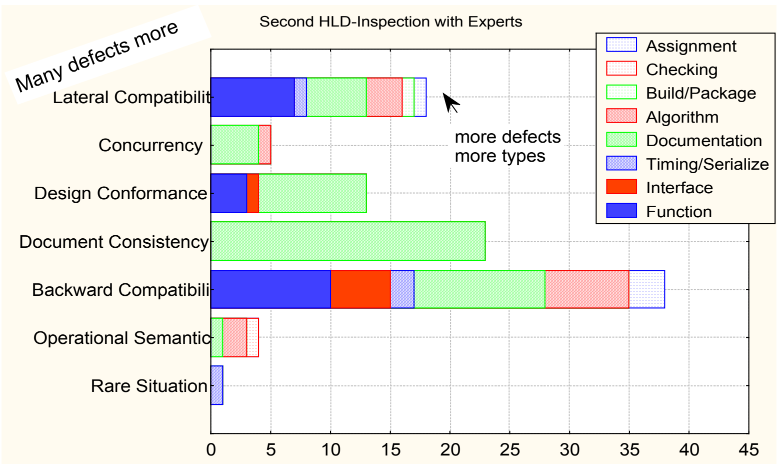

Figure 12 Data from second high-level-design inspection

This picture shows that many more defects have been found due to the second inspection, especially in the suspected compatibility areas. Moreover, defects covering more ODC defect types have been found indicating a more thorough inspection.

5.3.4 Evaluate and Improve Technologies

Defects can be used to evaluate technologies. Defect detection activities like inspections and testing can be evaluated and improved by investigating the kind of defects they find and do not find. Constructive activities can be evaluated and improved by investigating the defects that are typically generated by these technologies.

For example, in order to investigate reading techniques in the framework of inspections, the defects found by the reading technique and not found by the reading technique can be classified.

Defect classifications for such purposes can either be of a more general nature as presented in Section 3.2, or specifically tailored to the technology under study.

For example, in [Briand et al., 1998] it was the purpose to assess whether the introduction of inspections was beneficial to the project. One aspect of the benefit is that important defects are detected and not only clerical issues. For this purpose the severity of defects was classified with an ordinal scale (A being highest severity, C being lowest severity). The corresponding statistics are shown in Figure 13. Although the measurement goal was just to characterize the defects found, this result could be used to evaluate inspections since it showed that inspections reveal serious defects and not only clerical issues.

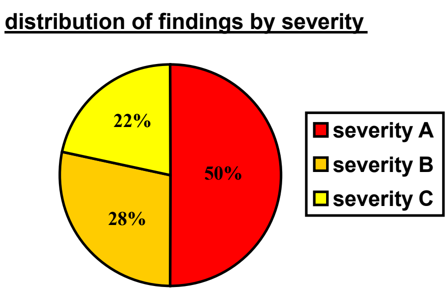

Figure 13 Severity of defects found during inspections

Another example of evaluating technologies with defect classification comes from the same source. Inspections are supposed to find defects before these can leak into subsequent development activities. For this purpose, in Figure 14 the defect origin is shown for each defect detection activity.

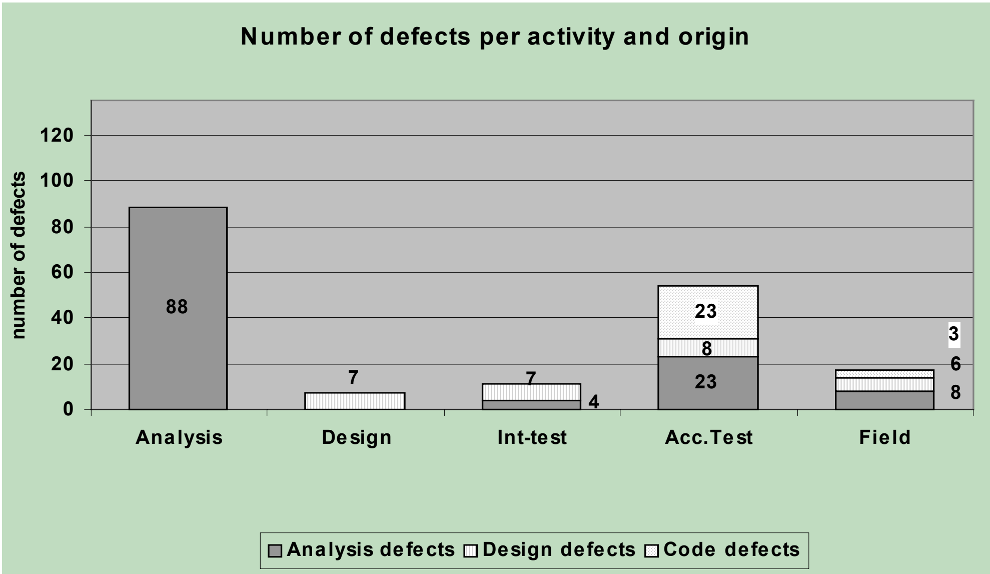

Figure 14 Defect Origin per Defect Detection Activity

This chart shows for each defect detection activity (analysis inspection, design inspection, integration test, acceptance test, field) the number of detected defects. In addition the number of defects is subdivided according to the different defect origins (Origin: Analysis, Design, Code).

This chart indeed demonstrates that many defects are detected much earlier in the life-cycle than without inspections. For example, without inspections, 95 defects more would have been necessary to be removed in testing.

5.3.5 Control Testing

As described in Section 3.2.3, the attribute Defect Type of ODC was designed to yield different signatures as the product passes through the life cycle so that each defect detection activity has its own characteristic expected distribution. Deviations of the actual distribution from the expected one can hint to potential problems in the project.

A typical example for this analysis is shown in Figure 15.

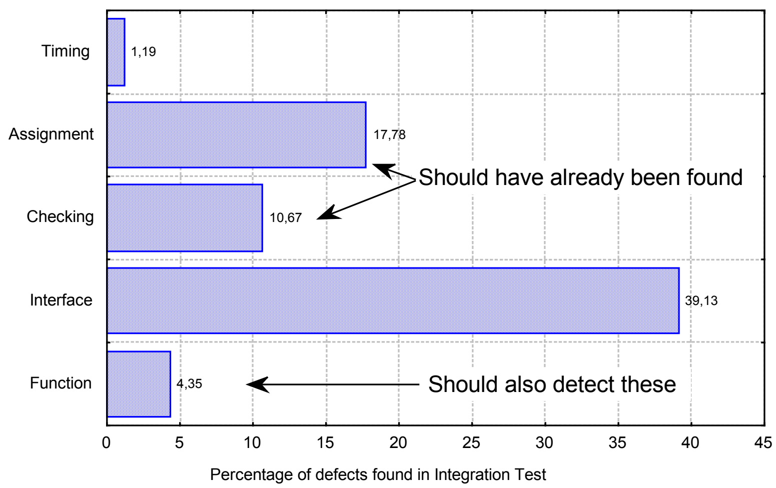

Figure 15 Defect type data

This figure shows the distribution of defects found in integration test. Expected to be found (in the environment under study) were Interface- and Function-defects. However, as it can be seen, only a small number of Function-defects have been found. On the other hand a large proportion of Assignment- and Checking-defects have been found. These are defects, however, are associated with unit test and should therefore have already been found. The interpretation of the development team here was that the product was prematurely in integration test. Thus, the defects that should have already been found hampered the integration test in finding Function-defects.

Such an analysis can be performed as regular entry criterion for test activities, as reported in [Hirsh et al., 1999].

5.3.6 Plan Testing

[Schultz99] describes a method to plan and control testing by using the ODC scheme. The rationale of this method is to map the number of defects found to the number of test cases and to analyze whether the defect space has been sufficiently exercised.

Using two attributes, namely Impact and Trigger, the defects detected during testing are classified and their cross-product determined as shown in Figure 16.

	Relia-bility	Perfor-mance	Inte-grity	Install-ability	Mig-ration	Stan-dards	Service-ability	Usa-ability	Cap-ability	Docu-mentation
Recovery	1					1		1	2	
Start / Restart									1	
Workload / Stress										
HW/SW config.									9	
Normal Mode	5	1		2					8	

Figure 16 Defects detected in testing classified by Impact and Trigger

As the ODC attribute trigger represented the reason, why a test case was written, the test cases can be also classified according to the trigger definitions. In addition, the test case is classified according to the impact that is most affected by the test. With these data from the classified test cases it is also possible to generate the cross-products as for the defects. Such a table is shown in Figure 17.

	Relia-bility	Perfor-mance	Inte-grity	Install-ability	Mig-ration	Stan-dards	Service-ability	Usa-ability	Cap-ability	Docu-mentation
Recovery										
Start / Restart				2					12	
Workload / Stress	4								2	
HW/SW config.			2						24	
Normal Mode			1			68			174	

Figure 17 Test cases classified by Impact and Trigger

Such a table can show, whether the entire defect space spanned by Impact and Trigger is exercised with test cases or whether only a part of the space is exercised.

In order to improve the test case planning, both tables can be combined as shown in Figure 18. In order to represent both sets of data in the same table, the number of test cases is coded by means of shading.

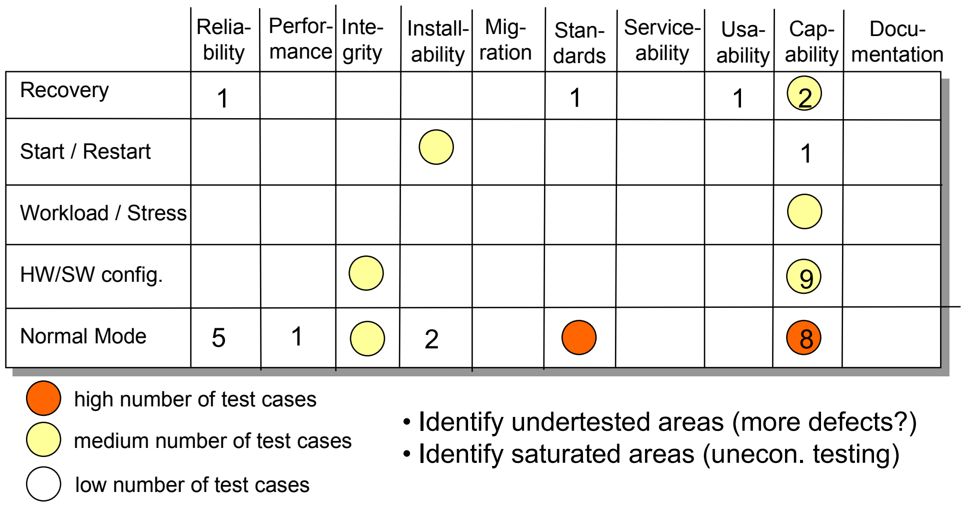

Figure 18 Combination of defects found and number of test cases

Based on the combination it is now possible to identify areas of potential improvement.

1. Areas with no tests and a low or high number of defects indicate the presence of defects that are not targeted by test cases. Here new test cases should be defined.

2. Areas with high numbers of tests and no defects indicate that you might spend too much effort on testing issues where no defects are. Here it should be checked whether it is from an economical more reasonable to spend some portion of this testing effort elsewhere.

3. Areas with a low number of test cases uncovering a large number of defects indicate a problem area. More testing should be done here or even to perform a further analysis of this part of the system (e.g., code inspections)

5.3.7 Reduce field defects

Defects are not only detected during development (i.e., by means of inspections and testing) but also after release to the customer. Analyzing these customer-found defects, which are here referred to as field defects, enable to find out why they were not detected during development. This knowledge can be used for improving the development and test process, and preventing similar field defects from occurring and slipping through to the customer again, thus increasing the quality of the system as perceived by the customer.

Therefore, the reduction of field defects is a special case of defect prevention as described in Section 5.3.2. For example, by means of histograms it is determined, what kind of defects are most bothering for the customer. These defects are the analyzed further.

One suitable way of focusing on customer-relevant defects is to use an attribute describing the defects' impact on the customer, such as the ODC attribute Impact. Such a distribution is shown in Figure 19.

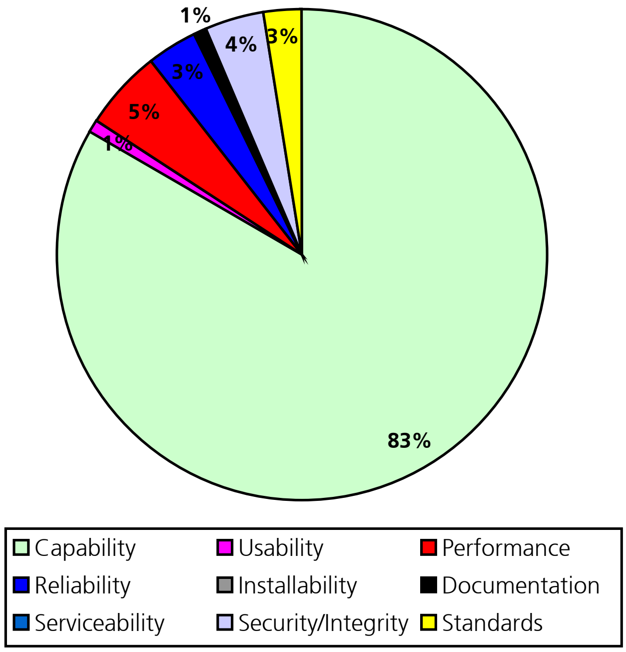

Figure 19 Impact distribution for field defects

In this example, process improvement efforts should focus on removing capability defects. A more detailed analysis of these defects is then to be performed.

One potential additional analysis is to compare the Impact of those defects detected in field with those detected during development (esp. testing). For example, the acceptance test or system test (depending on the concrete definitions of the test) should simulate the usage of the system from the users' point of view. Consequently the kind of defects detected in acceptance test should be similar to the kind of defects detected in field.

Thus, it should be analyzed whether the distributions are similar [Schultz99]. If necessary, a process change could be made to plan the test cases in order to cover the appropriate Impact-types for each function.

A second way to analyze field defect data is to use the ODC attribute Trigger [Bassin99]. This attribute describes for defects detected during development the catalysator that forces the defect to surface (e.g., the purpose of the test case). For defects detected in field the attribute value for the attribute Trigger is selected from the entire list of triggers which most closely matches the environment, special conditions, or catalyst which was required for the defect to surface. A trigger to activity mapping is then applied to determine – based on the Trigger-value – the activity in which the defect should have been detected.
An example of such data is shown in Figure 20.

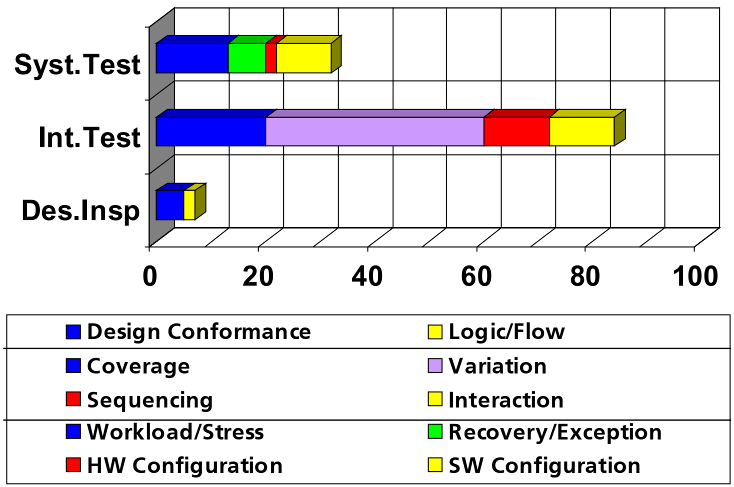

Figure 20 Trigger distribution for field defects

In such a graph it is visible, through which defect detection activity most of the field defects slip. This activity then has the largest potential for improvement. In addition, the Trigger-information offers a more detailed analysis: The defect detection activity can be specifically improved by paying more attention to those triggers that escaped into field.

For example, in Figure 20 the integration test can be improved by adding more test cases addressing Coverage and Variation. Thus, the rationale for the analyzed attribute Trigger can be used to obtain directly information on appropriate changes in the process.

A third way, how defect data has been analyzed within ODC are trigger profiles for field defects. For each set of triggers (i.e., system test triggers, etc.) the number and proportion of attribute values is plotted over time as shown in Figure 21.

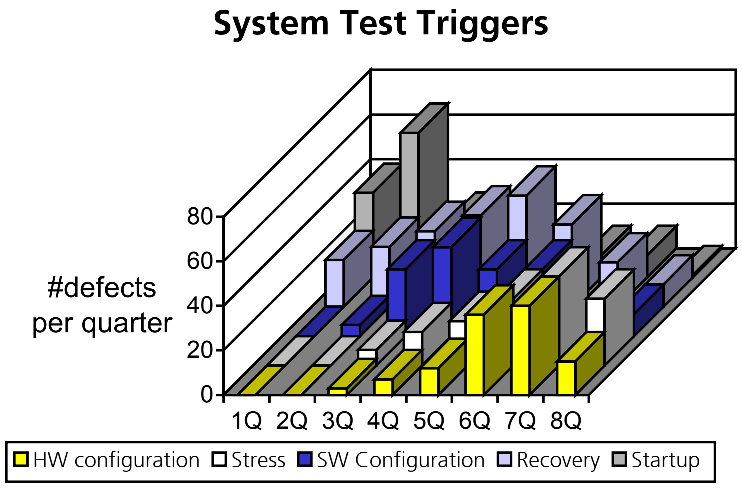

Figure 21 Trigger profile of system test triggers

Such a distribution is supposed to show how the customer used the software product. On one hand, this information can be used to plan testing activities. Those defects that are to be found early by the customer should consequently be also found early in test.

According to [Bassin and Santhanam, 1997], these profiles tend to be stable over multiple releases of the same product (not in terms of absolute numbers but in terms of proportions.). Therefore, these profiles can be used to predict to predict the number of field defects in the future. Such information can, for example, be used to estimate the required resources for service and support demands.

5.3.8 Monitor process changes

Often the distribution of defects tends to be fairly stable, especially when subsequent releases of the same product are developed. In these cases, the defect distribution can be used to monitor the impact of process changes.

For example, if a stable proportion of code defects (i.e., those to be found by code inspections) is found in system test, measures can be taken to find those defects already by means of code inspections (e.g., be developing a specific checklist). As a verification of the success of the process change “new checklist” can be verified as the proportion of code defects found in system test should decrease.

Such process changes can, for example, result as a basis from a defect prevention process as described in Section 5.3.2. For example, if in one release a systematic error has been found among Interface defects, and an effective action for preventing this error had been implemented, then the proportion of defects in this class should diminish over time [Card, 1998], [Messmore, 1999].

One example for such a baseline are the trigger profiles presented in Section 5.3.7. Since these profiles are stable over multiple releases, a successful process change should be visible after the first quarters after the next release [Bassin and Santhanam, 1997].

6 Summary and Open Issues

This report is based on a survey of papers and reports on defect classification. The main questions to be answered were "How can a defect classification be designed?" and "How can defect classification data be analyzed?"

In order to answer the first question, this report presented the aspects of a defect that have been measured in the literature, it presented the possible structures of a defect classification scheme, and it gave examples of frequently used defect classification schemes.

In order to answer the second question, this report presented general methods to analyze defect classification as reported in the literature as well as concrete analyses for a variety of measurement goals.

Yet, several questions are still open for defect classification:

One case study used defect classification (with ODC) to assess the effectiveness of inspections and control them appropriately. Although this approach has a large potential of enhancing inspections, few reports are available that report on replications of this type of analysis. Thus, to demonstrate whether this type of analysis is useful for analyzing inspection data in other environments, additional case studies should be performed (e.g., [Bridge and Miller, 1997])

A second point useful to analyze further is whether inspections and testing can be coordinated using defect classification data. When characterizing the defects found in inspections it should be possible to analyze the effectiveness of the inspection and make inferences about the required effectiveness of the testing process.

A third point is the relationship between measurement goals and defect classification attributes. The practitioner often wants to know what defect classification attributes have at least to be implemented in a given context to achieve a specific measurement goal. Although this report tried to address this issue by mapping attributes and analysis methods to measurement goals based on the existing literature, a more systematic and comprehensive approach is desirable.

7 References

Alexander Amezquita and Daniel P. Siewiorek, Orthogonal Defect Classification Applied to a Multidisciplinary Design, Tech. Rep. EDRC 05-100-96, Engineering Design Research Center, Carnegie Mellon University, 1996.

Basili, V. R., Green, S. ,Laitenberger, O., Lanubile, F., Shull, F., Sorumgard, S., Zelkowitz, M.V., "The Empirical Investigation of Perspective-Based Reading"; *Empirical Software Engineering – An International Journal,* vol. 1, no. 2, 1996.

Kathryn A. Bassin and P. Santhanam, Use of Software Triggers to Evaluate Software Process Effectiveness and Capture Customer Usage Profiles, in Proceedings of the the Eighth International Symposium on Software Reliability Engineering - Case Studies, pp. 103--114, 1997.

Kathryn A. Bassin, Theresa Kratschmer, and Padmanabhan Santhanam, Evaluating software development objectively, IEEE Software, vol. 15, pp. 66--74, Nov. 1998.

Inderpal Bhandari, Attribute focusing: machine-assisted knowledge discovery applied to software production process control, in Knowledge Discovery in Databases Workshop, pp. 61--69, 1993.

Inderpal Bhandari, Michael Halliday, Eric Tarver, David Brown, Jarir Chaar, and Ram Chillarege, A case study of software process improvement during development, IEEE Transactions on Software Engineering, vol. 19, pp. 1157--1170, Dec. 1993.

Inderpal Bhandari, Bonnie Ray, Man-Yuen Wong, David Choi, Akemi Watanabe, Ram Chillarege, Michael Halliday, Alan Dooley, and Jarir Chaar, An inference structure for process feedback: technique and implemenation, Software Quality Journal, pp. 167--189, 1994.

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.

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.

Norm Bridge and Corinne Miller, Orthogonal Defect Classification -- Using Defect Data to Improve Software Development, in Proceedings of the 7th International Conference on Software Quality (7ICSQ), 1997.

Harald Buczilowski, Defect Reduction in VSE -- Methodology and Results, in Proceedings of Sixth International Symposium on Software Reliability Engineering, pp. 274--277, Oct. 1995.

David N. Card, Learning from our Mistakes with Defect Causal Analysis, IEEE Software, pp. 56--63, Jan. 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.

Ram Chillarege and Kathryn A. Bassin, Software Triggers as a function of time -- ODC on field faults, in Fifth IFIP Working Conference on Dependable Computing for Crititical Applications, 1995.

Siddhartha Dalal, Michael Hamada, Paul Matthews, and Gardner Patton, Using Defect Patterns to Uncover Opportunities for Improvement, in Proceedings of the International Conference on Applications of Software Measurement, 1999.

Khaled El Emam and Isabella Wieczorek, The repeatability of code defect classifications, in Proceedings of the Ninth International Symposium on Software Reliability Engineering, pp. 322-333, 1998.

Norman E. Fenton, Software Metrics - A Rigorous Approach. Van Nostrand Reinhold, 1991.

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

William A. Florac, Software Quality Measurement: A Framework for Counting

Problems and Defects, Tech. Rep. CMU/SEI-TR-22, Software Engineering Institute, Sept. 1992.

Michael Fredericks and Victor Basili, Using Defect Tracking and Analysis to Improve Software Quality, Tech. Rep. DACS-SOAR-98-2, DoD Data Analysis Center for Software (DACS), 1998.

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.

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

Michael Halliday, Inderpal Bhandari, Jarir Chaar, and Ram Chillarege, Experiences in Transferring a Software Process Improvement Methodology to Production Laboratories, Journal of Systems and Software, vol. 26, no. 1, pp. 61--68, 1994.

Barbara Hirsh, Robert Bleck, and Steven Wood, A Comprehensive Defect Prevention Program Using the ODC Methodology, in Proceedings of the 10th International Symposium on Software Reliability Engineering (Fast Abstracts and Industrial Practices), pp. 61--81, 1999.

Jon T. Huber, Software Defect Analysis: Real World Testing Implications A Simple Model for Test Process Defect Analysis, in Proceedings of the 2nd International Quality Week Europe, (625 Third Street, San Francisco, CA 94107-1997, USA), Software Research Institute, 1998.

Watts S. Humphrey, A Discipline for Software Engineering. SEI Series in Software Engineering, Addison-Wesley Publishing Company, 1995.

IBM, Center for Software Engineering
http://www.research.ibm.com/softeng/ODC/ODC.HTM

Institute of Electrical and Electronics Engineers, IEEE Standard Classification for Software Anomalies, IEEE Std. 1044-1993, 1994.

Robert Bruce Kelsey, Integrating a Defect Typology with Containment Metrics, ACM Software Engineering Notes, vol. 22, pp. 64--67, Mar. 1997.

Christian Kuhröber, Isabellea Wieczorek (supervisor), Dieter Rombach (supervisor) ``Literature Survey on Defect Classification.'' Project Thesis, Fraunhofer IESE, Sauerwiesen 6, D-67661 Kaiserslautern, Germany, 1997.

Marek Leszak, Dewayne E. Perry, and Dieter Stoll, A case study in root cause defect analysis, in Proceedings of the 22nd International Conference on Software Engineering, pp. 428-437, 2000.

R.G. Mays, C.L. Jones, G.J. Holloway, and D.P. Studinski, Experiences with defect prevention, IBM Systems Journal, vol. 29, no. 1, pp. 4--32, 1990.

Peter Mellor, Failures, faults and changes in dependability measurement, Information and Software Technology, vol. 34, pp. 640--654, Oct. 1992.

James Miller, Marc Roper, and Murray Wood, Defect Classification for Object-Oriented Maintenance, in Proceedings of the 2nd International Workshop on Empirical Studies of Software Maintenance, 1997, pp. 15--19, 1997.

Thomas J. Ostrand and Elaine J. Weyuker, Collecting and Categorizing Software Error Data in an Industrial Environment, Journal of Systems and Software, vol. 4, pp. 289--300, 1984.

Mark C. Paulk, Dennis Goldenson, and David M. White, The 1999 Survey of High Maturity Organisations, Tech. Rep. CMU/SEI-2000-SR-002, Software Engineering Institute, Feb. 2000.

Shari Lawrence Pfleeger, Software Engineering: Theory and Practice. Prentice-Hall,Inc, 1998.

Adam Porter, Larry G. Votta, Victor R. Basili, Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment, IEEE Transactions on Software Engineering, 1996, pp 563-575.

David Rentschler, Implementing Orthogonal Defect Classification , Transactions from the Fifth International Conference on Software Quality, October 1995, pages 277-279.

Charles P. Schultz, Orthogonal Defect Classification (ODC) Based Test Planning and Development, in Proceedings of the International Conference on Applications of Software Measurement, 1999.

Shull, F. Travassos, G.H., Carver, J., and Basili, V.R. "Evolving a Set of Techniques for OO Inspections." University of Maryland Technical Report CS-TR-4070, October 1999.

Jack Silberman, Robot Orthogonal Defect Classification -- Towards an In-Process Measurement System for Mobile Robot Development. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Jan. 1998.

N.B. Sreenivasan, Experiences with Orthogonal Defect Classification Technique, in Proceedings of the 10th International Symposium on Software Reliability Engineering (Fast Abstracts and Industrial Practices), pp. 82--87, 1999.

Document Information

Title:	Developing and Using Defect Classification Schemes
Date:	September 1, 2001
Report:	IESE-072.01/E
Status:	Final
Distribution:	Public