An Industrial Case Study of Implementing and Validating Defect Classification for Process Improvement and Quality Management

Bernd Freimut
Fraunhofer IESE
Sauerwiesen 6
67661 Kaiserslautern, Germany
freimut@iese.fhg.de

Christian Denger
Fraunhofer IESE
Sauerwiesen 6
67661 Kaiserslautern, Germany
denger@iese.fhg.de

Markus Ketterer
Bosch GS-EC/ESP
P.O. Box 30 02 40
70442 Stuttgart, Germany
Markus.Ketterer@de.bosch.com

Abstract. *Defect measurement plays a crucial role when assessing quality assurance processes such as inspections and testing. To systematically combine these processes in the context of an integrated quality assurance strategy, measurement must provide empirical evidence on how effective these processes are and which types of defects are detected by which quality assurance process. Typically, defect classification schemes, such as ODC or the Hewlett-Packard scheme, are used to measure defects for this purpose. However, we found it difficult to transfer existing schemes to an embedded software context, where specific document- and defect types have to be considered. This paper presents an approach to define, introduce, and validate a customized defect classification scheme that considers the specifics of an industrial environment. The core of the approach is to combine the software engineering know-how of measurement experts and the domain know-how of developers. In addition to the approach, we present the results and experiences of using the approach in an industrial setting. The results indicate that our approach results in a defect classification scheme that allows classifying defects with good reliability, that allows identifying process improvement actions, and that can serve as a baseline for evaluating the impact of process improvements*.

1 Motivation

In embedded systems, software allows satisfying the need of customers for new and unique functionality. Especially in the automotive domain, this increasing importance of software increases product complexity, so that consequently, software development becomes a more and more complex task. In order to maintain and guarantee a high quality level in the final product, the application of quality assurance techniques, such as software inspection and testing, is therefore of crucial importance. Thus, the impact of these techniques should be optimized and the individual techniques should be integrated into a most effective and efficient combined inspection and testing strategy.

However, experience indicates that different quality assurance techniques are used mainly independently from each other [3]. Moreover, it is often not clear how individual techniques contribute to the overall system quality, i.e., there is no or little knowledge regarding the quality assurance capability of the applied techniques. Without such information, the development of an integrated quality assurance strategy guaranteeing a stable level of quality is difficult, if not completely impossible.

To overcome this lack of information, the introduction and continuous application of measurement is one possible solution. Measurement is used to provide relevant information for quality management needs, such as the identification of defect detection activities that many (types of) defects slip through or the identification of techniques with high defect detection potential.

Defects carry a lot of information that can be measured and analyzed with respect to the objectives of a quality assurance strategy. Thus, the central element of a quality measurement program is a useful and easily applicable defect classification scheme. There are a variety of defect classification schemes that have been successfully used for different measurement purposes. The most prominent ones are the Hewlett-Packard Scheme [13] and the Orthogonal Defect Classification (ODC) scheme [7].

However, even though these schemes are successful in their specific context, it seems to be difficult to customize them to the context of other companies. This is especially true if the type of documents and considered defects differ from traditional ones. Therefore, an approach is warranted that customizes defect classification schemes and their measurement goals to such special environments.

In this paper we present such an approach that we developed to define, introduce, and validate a customized defect classification scheme at Robert Bosch GmbH in the business unit for Gasoline Systems (GS). The core of the approach is the combination of (1) measurement know-how regarding the principles of good defect classification schemes (e.g., successful elements of existing schemes) and (2) domain know-how about essential defect information in the special context of developing automotive embedded systems. Since the expertise for each type of know how differs with different roles, the core aspect of our approach is a three-step strategy implementing this combination. The objective of the approach is the development of a defect flow model that comprises a customized defect classification scheme. In

11th IEEE International Software Metrics Symposium (METRICS 2005)
1530-1435/05 $20.00 © 2005 **IEEE**

each step of the process, interviews with domain experts play a crucial role to successfully customize best practices of existing schemes to the company's context.

In the remainder of the paper, we present the context of the business unit of Bosch GS in Section 2. In Section 3, related work on defect classification schemes is presented. The measurement objectives of Bosch GS are discussed in Section 4. Our customization approach is described in Section 5 and its application in a case study is discussed in Section 6. The evaluation results of the case study are presented in Section 7. Section 8 concludes the paper and gives an outlook on future work.

2 Company Context

One of the products developed by the business unit Gasoline Systems (GS) of Robert Bosch GmbH are electronic control units for gasoline engines. In these systems, embedded software is the decisive component, for which roughly 900 employees in software development are responsible.

The development process in which the customization approach will be applied is shown in Figure 1:

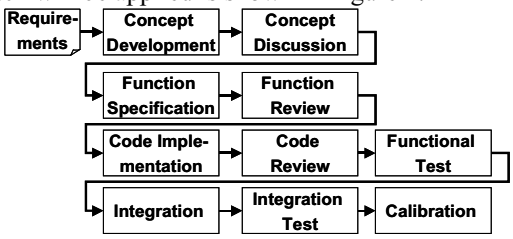

Figure 1: Development Process

Based on the requirements, a physical concept is developed that is specified in a function specification. This specification is implemented in software and the resulting code modules are integrated. Finally, during calibration, necessary parameters are set for a specific application. Quality assurance is performed throughout the process.

To achieve high product quality despite increasing product complexity in terms of functionality and number of SW variants, the software development process has been continuously improved during the past years using a CMMi-based approach. Within this context, GS perceives measurement as a key factor to successfully identify and introduce software process improvements.

Setting the focus on establishing quantitative data management to optimize the quality assurance strategy requires a measurement instrument that allows the identification of strengths and weaknesses of the current quality assurance activities throughout the whole development life cycle. When implementing process improvements based on quantitative data, this instrument has to be able to validate the effects of these process changes.

The leading question in optimizing the quality strategy is which kinds of defects should be addressed by which technique. As an answer, the measurement instrument has to provide information on the question of "Which defects are introduced in which development activity, and with which quality assurance technique are they detected and resolved?"

The initial step to answer this central question is to build up a measurement baseline characterizing the current quality assurance process within the development life cycle. The foundation of this measurement program is a customized defect classification scheme. The scheme can be used to answer the question of which types of defects are introduced and resolved in which development phases. Furthermore, it ensures that all people involved in the development process and the measurement program gain a common understanding of relevant defects.

3 Related Work in Defect Measurement

Defect measurement plays an important role in most software measurement programs ([12], [9], [13].) This importance stems from the fact that defects are a direct indicator of software quality, and that most software process improvement activities either aim at finding more defects or at finding defects earlier less expensively, or aim at injecting fewer defects.

The most basic metric is the number of defects detected by a certain V&V activity. Determining the number of defects that is detected during the entire life-cycle provides a defect profile [5], [9]. This defect profile forms a baseline for the quality assurance process. Any changes to this process can be observed as changes to this baseline. For example, introducing requirements and design inspections will cause more defects to be detected early in the life cycle [12].

The next metric that is typically used is the *origin* or *injection phase* of a defect, where a developer classifies in which phase or activity the defect was introduced into the system. With this information it is possible to determine the number of defects that were introduced in a given phase or activity, and it is possible to determine the effectiveness of quality assurance techniques by relating the number of defects found to the number of defects not found (e.g., as defect removal effectiveness or yield [15]).

Classifying the *origin* of a defect is one example of defect classification in general. A defect classification scheme contains one or more defect classification attributes that capture various aspects of a defect. For example, [18] proposes a framework of eight high-level key attributes that capture different defect aspects. Each of these defect classification attributes is measured on a nominal or ordinal scale with a set of pre-defined values, the so-called attribute values. The challenge in designing a defect classification is to select appropriate attributes and corresponding attribute values.

Researchers used defect classification schemes to characterize the defect detection ability of quality assurance techniques, for example reading techniques [21] and automated defect detection techniques [19], or to describe defects found in particular kinds of documents such as use cases [1], [8].

The most often referenced defect classification schemes from an industrial point of view are the HP Scheme [13], which aims at deriving process improvement proposals, and the ODC Scheme from IBM [7], which also aims at controlling the progress of a development project.

The HP Scheme is shown in Figure 2. Once a defect is detected, the developer determines the activity in which the defect was introduced and assigns an appropriate attribute value of the attribute *origin*. Next, the attribute *type* describes the defect of a particular origin in more detail. Finally, the attribute *mode* describes why the defect was a defect.

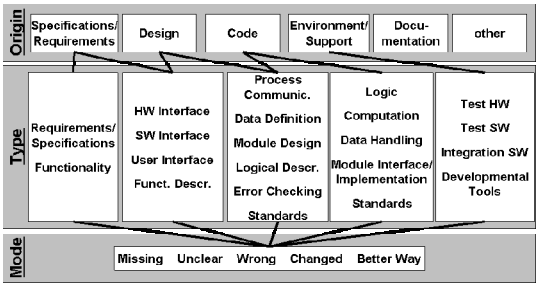

Figure 2: HP Scheme

In the ODC scheme, which has evolved as a quasi-standard in industry and research, several useful ideas are combined. Orthogonal aspects of a defect are to be captured in distinct defect attributes. Thus, the entire IBM scheme contains eight different attributes, which capture product-related, development-related, and defect-detection-related information. Of these eight attributes, the attribute used most often is the *defect type,* which we focus on here.

The *defect type* captures the nature of the fix, i.e., it addresses the question: "What did the developer correct in order to resolve the defect?" The ODC inventors emphasized that the set of attribute values of the defect type should match to the set of correction *activities* the developer performs. With this rationale, the classification process is easier for the developer, since he or she can almost decide objectively, based on the activity performed, which attribute value to assign.

In addition, they designed the attribute values in such a way that certain defect types are addressed by specific defect detection techniques. Thus they achieved a scheme in which certain attribute values peak for certain detection activities. Consequently, for each defect detection activity, an expectation exists of what the *defect type* distribution should look like. This baseline can be used to control a project by comparing the expectation to the actual situation [6][7]. It has also been used for quality management purposes [17]: Based on a baseline, an improvement objective can be set to reduce or increase the proportion of a particular defect type in a specific defect detection activity. Due to the detail of the attribute values, it is then straightforward to target corresponding improvement actions. In addition to these applications, ODC has been used for additional measurement purposes such as identifying process improvements, improving products, and focusing testing efforts [22][2].

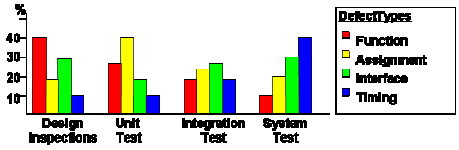

Figure 3: ODC defect profile

In industry, self-developed schemes are also used. But such schemes often have deficiencies: Typical problems are that schemes are non-orthogonal (i.e., more than one attribute value can be selected) [20], that the meaning of attribute values is unclear (e.g., "to me everything is a *logic* defect"), or inconclusive analyses.

Thus, our conclusion is that there are proven defect classification schemes that can be used when introducing defect measurement. However, it is not possible to directly transfer these schemes, as they are meant for certain contexts. For example, the ODC *defect type* addresses mainly defects found in code; thus the scheme is hard to use for early phases such as requirements and high-level design. Moreover, it is crucial to consider document types, as these can vary depending on the domain. For example, in our context, the Function Specification differs from typical design documents.

Therefore, we believe that a general approach on how to define and customize a good classification scheme is necessary and will be a beneficial contribution both from a research and from an industrial point of view. In the context of our work, such an approach is required to develop a customized defect classification scheme.

4 Measurement Objectives

Considering the overall objective of GS, namely to establish a measurement instrument for the quality assurance process with the intention of establishing quantitative management of the combined inspection and testing strategy (based on the question of what type of defects should be addressed by which defect detection activity) the issue is which of the aspects presented in the related work need to be considered.

The survey of the state of the art reveals several valuable elements of defect classification schemes. First of all, we deem it necessary to capture the number of defects per defect detection activity and the origin of a

defect (i.e., the activity in which it was injected), since this is basic information. This information can be synthesized as a defect flow model as shown in Figure 4. On the positive y-axis, the defect flow model provides information on how many defects are introduced into the development cycle in which development steps. On the negative y-axis, the amount of defects that are detected in a defect detection activity are shown. Thus, a defect flow model provides useful information on the life cycle of a defect and represents a first measurement baseline.

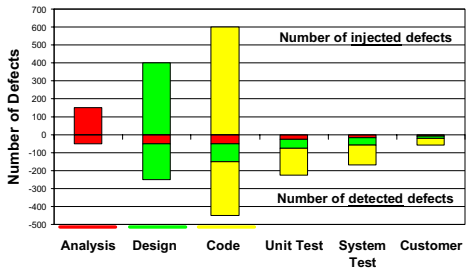

Figure 4: Defect Flow Model

To fulfill the objective of creating an integrated quality strategy, it is necessary to enhance the defect flow model with defect information on a more detailed level. Attributes like the HP *type* or the ODC *defect type* will provide valuable analyses for a quality strategy, as these attributes characterize a defect at a more detailed level of granularity. Based on such detailed information it is possible to determine, which quality assurance techniques can address which defect types.

Similar to Figure 3, it is therefore our objective to capture the signature of such a defect type over the development process. Thus, we target a representation as shown in Figure 5. Each diagram represents a defect detection activity. On the y-axis the defect type distribution shown. This information enables us to evaluate which types of defects are detected in which defect detection activities.

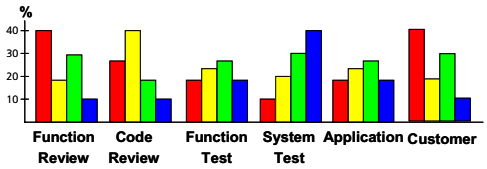

Figure 5: Signature over process

To ensure that these analyses deliver reasonable and useful results, it is essential for the different elements used in the analyses to be well defined and understood by all persons involved. This concerns, in particular, the underlying defect classification scheme, which here consists of the attribute *detection* capturing the detection activity revealing the defect, the attribute *injection* capturing the activity injecting the defect, and an attribute *type* according to the ODC defect type.

This scheme needs to be well defined and should fulfill the quality properties of a good defect classification scheme [11]: The attributes of the scheme need a *clear meaning and definition*. This definition has to be developed with all stakeholders who have to use the attributes and need an understanding of the attribute. In addition, the *values of the different attributes also need a clear definition*. To facilitate the understanding of the attribute values, an *example should be given for each value*. This example should be taken from the company's context. The classification scheme has to be *complete and orthogonal*. "Complete" means that it is possible to classify each defect with the scheme. Orthogonal means that only one attribute value characterizes the defect (it is not possible to select two attribute values for one defect). Finally, the scheme should contain a *small number of attribute values,* as too large a number can make the selection of the appropriate attribute value difficult und therefore unreliable. A recommended value is between 5 and 9 attribute values, as this represents the number of items that human short-term memory can keep [7]. Finally, the scheme has to allow *useful analyses* and provide insightful information into the V&V process.

5 A Process for Defining and Introducing Defect Classification Schemes

5.1 Process Design Considerations

We considered two issues to be vital for the successful introduction of defect classification into an industrial environment. The first issue addresses the management point of view with the question of how to motivate and visualize the benefits of defect classification. This is an important aspect of change management, since an understanding of the benefit is essential for management commitment and data quality.

To demonstrate the potential of a detailed investigation into defect classification and thus enable management decisions, we aimed at estimating the current defect flow within the organization using available data and expert estimates. Using only available data and expert estimates, such an estimate can be obtained rather quickly and does not require the costly implementation of defect metrics. Simultaneously, it allows an initial insight into the defect flow and the general strength and weaknesses of the V&V process. Our experience with such an approach in industrial environments is rather good [4].

The second issue for the successful introduction of defect classification addresses the technical question of how to define a good defect classification scheme. To do so we made the following decisions:

First, a good defect classification scheme requires input both from measurement experts and from domain experts. The input of domain experts is necessary as they have the expertise and experience regarding what

constitutes a defect in the environment under study. The input of measurement experts is necessary as they bring in the expertise and experience regarding the structure, properties, and analysis of defect classification schemes. Therefore, our proposed process is based on interviews that are prepared by measurement experts and in which the knowledge and experience of domain experts is captured.

Second, we followed the ODC idea to capture the question "*What did you fix in order to correct the defect?*" with a defect classification attribute. This approach promised to achieve good results, as developers can classify defects according to the activity they perform. Moreover, it provides a clear focus for the interviews.

5.2 Process Overview

Our process consists of three phases shown in Table 1. The first phase is the basic model definition phase. The objective of this phase is to define the defect flow model by defining the defect attributes *injection* and *detection,* which is usually done easily. A second objective is to motivate the collection of defect classification data for defect types. This motivation is supported by two means: First, an estimation of the current defect flow. Using such estimates, a first characterization of the V&V process can be obtained. Second, by eliciting an initial version of the empirical QA strategy that characterizes what types of defects can be detected well by a given detection activity and what types of defects slip through. By presenting such information, the definition of more detailed defect classification can be motivated more easily. Moreover, these activities are mainly interview-based and therefore less cost-intensive than actual data collection.

The objective of the second phase is to define more detailed defect classification attributes. Here, a suitable set of defect attributes is selected, which contains the *Correction Type* (the equivalent of the ODC defect type) and others as necessary. Interviews are performed by measurement experts to capture the domain know-how of what a defect is and how it is corrected.

The objective of the third phase is to implement the scheme, collect data according to the scheme, assess the quality of the scheme, and use it.

Phase	Activity
Basic Model Definition	Develop Defect Classification Attributes for Defect Flow Model (i.e., attributes: *injection*, *detection*)
	Determine Defect Flow based on data and estimates
	Determine Qualitative QA-Strategy
Detailed Model Definition	Select appropriate defect classification attributes (i.e., attribute *Correction Type* and others)
	Determine Attribute Values
	Determine Analysis Charts
Pilot Application	Train Developers in Defect Classification
	Implement Data Collection Procedures
	Check Data Quality
	Check Meaningfulness of Analyses

Table 1: Process Overview

In the following subsection we will discuss each of these phases in more detail.

5.3 Basic Model Definition

This phase contains the activities Develop Defect Flow Model, Determine Defect Flow Estimate, and Determine Qualitative QA Strategy.

Develop Defect Flow Model

The first step is to develop a defect flow model, which is basically the definition of a defect classification scheme consisting of two attributes: *detection* (i.e., in which activity was the defect detected) and *injection* (i.e., in which phase was the defect injected). An initial version of these attributes can be straightforwardly derived from the company's process model: V&V processes determine the *detection* attribute values, while constructive activities determine the *injection* attribute values. Here it is necessary to balance the details of the defect classification and the number of attribute values. Moreover, it is necessary to identify those phases from which no defect data can be collected. A typical case is, for example, a debugging activity performed by the authors themselves. Introducing new data collection procedures here would only hamper defect reporting. In this case the boundaries of a process step have to be defined accordingly.

An important step is then to check the applicability of the resulting scheme with the developers. In structured interviews, the purpose of the scheme is presented, and feedback for missing attribute values (defect sources or detection activities) is obtained. Particularly important is the applicability of the *injection* classes: Is it possible to assign actual defects to these attribute values? Are there any *injection* sources that have been overlooked? Based on this feedback, the scheme is then revised.

Develop Defect Flow Estimate

Based on the defined defect flow model, we aim at giving management an initial version of what the defect flow will look like. Since defect counts from inspections and testing (i.e., *detection* information) are typically available in industrial contexts but data on defect origins (i.e., *injection*) is not [16], we attempt to combine project data and expert opinion to estimate the defect flow, following the approach presented in [4].

This is done via the following steps: First, the required data are determined. This is the number of defects $n_{d,o}$ of origin o detected in detection activity d for all o and d. Second, it is determined which data are already available and which not. If data is not available, the model is reformulated accordingly. Typically, the number of defects n_d is available but without *origin* information. In this case the model is reformulated as $n_{d,o}=n_d*perc_{d,o}$, where $perc_{d,o}$ is the percentage of defects of origin o in d. This quantity is obtained through expert opinion. In a third step, interviews are carefully prepared and

conducted, where the developers estimate the missing quantities. In order to explicitly model the uncertainty of the estimates, we elicit triangular probability distributions [24]. Finally, the probability distributions and project data are combined using MC-Simulation [24] to obtain the resulting defect flow (cf. Figure 6).

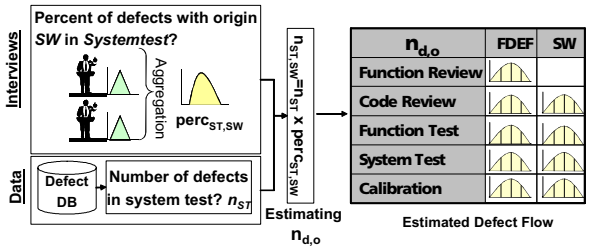

Figure 6: Approach to determine prototypical profile

Since the expert estimates exhibit uncertainty, so does the resulting defect flow.

Determine Qualitative QA-Strategy

We understand a qualitative QA strategy as a listing of defect classes that are either easy or hard to detect by each defect detection phase, as shown in Table 2.

Easy to detect				
		Type 1		Type 2
	Design Inspection	**Code Inspection**	**Unit Test**	**System Test**
Hard to detect	Type 1			
	Type 2			

Table 2 Qualitative QA Strategy

Such a table is filled in with developers, where, on purpose, we give no definition of what we mean by "defect type", so that developers have to use their own, domain-specific view. This allows the measurement experts to learn the language of the developers, i.e., what is a defect "type" for them. Moreover, such a table is a first visualization of the empirical QA strategy and can even identify improvement potential. Consequently, it can motivate the implementation of a more detailed defect classification.

5.4 Detailed Model Definition

In this phase of the process, the measurement team selects the attributes and their values, which are used to enhance injection and detection.

The core activity here is to define the *Correction Type* that captures the question "What is fixed in order to correct the defect?". We chose this attribute as a central element of the classification scheme since we learned that practitioners think and talk about technical defects of certain documents. The correction type captures exactly this knowledge. In consequence, the attribute values of the correction type represent domain knowledge and are easy to understand, as they represent the "natural" way of thinking about defects.

In addition to this attribute, it might be necessary to identify additional attributes that are useful for analyzing subparts of the data. For example, it might be useful to classify properties of the product. In a reuse-centered development, we might distinguish between software that is written for reuse and software that is written for individual projects. We might distinguish between code modules written manually or generated automatically.

While these additional attributes should be independent, we suggest relating *injection* and *correction type* as in the HP scheme (cf. Figure 2): First, select the *injection* that characterizes the activity that injected the defects. Then, the *correction type* characterizes the defect in more detail. Since each injection activity typically uses a document type of its own, we develop an attribute correction type for each attribute value of the *injection* attribute.

Since in our context we deal with document types for which the existing design/code attribute values of the ODC defect type cannot be used, we need to define new attribute values. To define these attribute values, domain knowledge is necessary, since this knowledge does not only identify syntactic structures, but also the semantic structures that a document is composed of. It also ensures that the defect classes match the context at hand.

Thus, we first perform structured interviews with a small group of developers who deal with the document type considered. In these interviews we present the overall objective of the measurement system, the specific meaning and function of the attribute discussed, and the question it is supposed to answer. The developers then brainstorm typical defect correction types, which the measurement team tries to formulate as attribute values.

Next, the initial set of attribute values is consolidated from a larger set of developers using a workshop or questionnaire. Here, we focus on issues such as completeness of the attribute values, understandability of the definitions and examples, as well as the orthogonality. To check the latter, we prompt the participant to consider pairs of attribute values and think of defects that could match both. If this is the case, the definitions have to be clarified or the defect classes need to be merged.

5.5 Pilot Application

In the pilot application phase, the defined defect classification scheme is implemented and data are collected. These activities follow the processes described in the measurement literature [23].

Here, particular emphasis is placed on educating the developers in using the classification scheme. Using the definition and examples defined in the previous step, this is straightforward. Another issue is to provide support for collecting measurement data, which might be paper-based or tool-based. In the latter case it is possible to augment data collection sheets with attribute definitions and

examples [14]. Finally, an important issue is to observe any difficulties the developers have in assigning defects to attributes and attribute values. Here the measurement team checks whether the classified defects are in accordance with their definitions. A quantitative method to check the quality of the classification in terms of its reliability is advisable. This method is presented in the next section.

6 Case Study at Bosch GS

We followed the process proposed at Bosch GS. In this section we report on some of our results and experiences with the process in particular and with defect classification in general. However, since defect information from any company is a highly sensitive issue, we can only show selected results due to confidentiality reasons.

6.1 Basic Model Definition

Develop Defect Flow Model

To develop an initial set of attribute values of the *detection* attribute, we started with those activities that already collect defect data or that are important defect detection activities: {function review, code review, function test, system test, calibration}. The feedback we received from the interviews was that in integration, which per definition is a constructive activity, defects can also be uncovered and this activity is therefore missing in our initial set. Moreover, several additional activities were mentioned that are also supposed to detect defects. For example, during function development, simulation can be performed, which can reveal defects. However, since it would be difficult to start data collection in these processes, we decided to aggregate these processes with their constructive counterpart. Thus, we counted simulation as part of function development and started counting function defects when the Function Specification was submitted for inspection.

For *injection* we started with {concept development, function spec, software coding} as an analogy to the generic {analysis, design, and code}. The interviews revealed, however, that it is hard to identify concept defects. This is due to the fact that concept development, due to the SW architecture, also comprises essential steps of function development. Consequently, we changed the attributes to {function spec, software coding}. This had the additional benefit of the developers already distinguishing defects between function spec and software coding. Since our scheme was to capture the actual view and defect vocabulary in place, we considered this an important issue to address.

Develop Defect Flow Estimate

To develop the defect flow estimate, we had one V&V process for which we had both the number of defects and well as the *injection* classification. For most inspection and testing processes, the number of defects was available. In order to obtain the required information about the *injection* classification through expert opinion for these processes we asked the corresponding developers which percentage of the defects was injected in the function specification and which percentage in software.

For one testing process, no systematic storage of defect numbers or classification was available. For this process we thus estimated the total number of defects. Since our objective was to elicit only estimates of quantities that a developer can easily relate to, we asked for the number of tests, which was available as project data, and had testers estimate the number of defects detected per test.

Applying the process described before, the resulting distribution was generated as shown in Figure 7, which shows only exemplary data due to confidentiality reasons. This figure shows the number of injected defects on the positive y-axis. The bars denote minimum, maximum, and most-likely value of the number. The positive y-axis shows the percentage of function spec and software defects. The negative y-axis shows the percentage of total defects found for each V&V activity. Similar graphs were generated considering FDEF and code defects separately.

Despite the fact that estimates are subject to uncertainty, we were able to draw several conclusions from them. First, the majority of defects are detected before the system test, an observation confirmed by developers. Second, we concluded that code reviews showed rather high effectiveness, a fact also confirmed by the developers.

The benefit of these charts was that quantitative feedback about the V&V process can be provided, which illustrates the potential and benefit of such a model. It also indicates that for the overall objective, a more refined defect classification was desirable, as the distinction between function specification and software is too coarsely grained to answer the question of which type of defects should be detected by which defect detection activity.

Determine Qualitative QA-Strategy

In the same interviews in which we elicited the estimates we also asked the developers to fill in a table such as the one shown in Table 2. The application of this table was quite effective, since several conclusions could be drawn from it: First, in combination with the basic defect flow model, there was a clear motivation that for our purpose, more detailed defect classes are necessary. Consequently, management could be convinced easily to invest in implementing defect classification.

Moreover, we were able to identify one defect class that was introduced early in the process but detected rather late. Here, we not only identified improvement

potential, but also motivated the explicit consideration of an empiric QA Strategy.

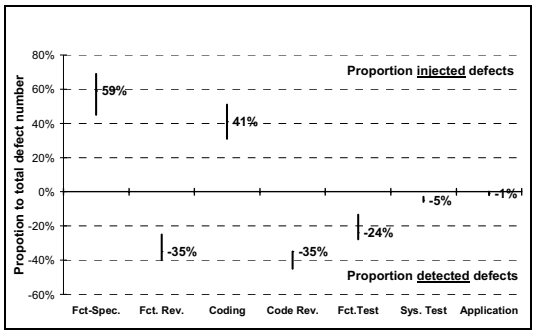

Figure 7: Defect Flow Estimate

6.2 Detailed Model Definition

To structure the detailed scheme, we borrowed from the HP scheme: First, the developer selects the defect *injection* between Function Specification and Code, which describes the faulty document type. Then a correction type was to be defined for each document type. (cf. Figure 8; due to confidentiality, only an excerpt is shown).

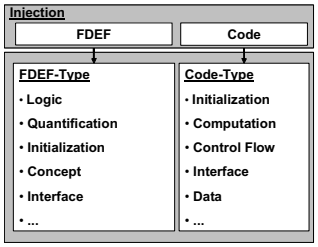

Figure 8: Defect Classification Architecture

This structure has several advantages in the context at hand: First, it matches the defect correction process: the Function Developer decides whether the defect is in the FDEF or the Code and routes the defect accordingly. In this phase, the *injection* is collected. After the corresponding developer corrects the fix, he or she collects the *Correction Type*. Second, the *Correction Type for FDEF* documents is only relevant for Function Developers while the *Correction-Type for Code* is only relevant for SW developers. Since the two processes are organizationally separated, so should be the responsibility for the corresponding defect attributes.

For SW documents, a defect classification already existed, which could be easily integrated as Correction-Type. For FDEF documents, we had to design the Correction Type. Here we had an initial interview with Function Developers in which an initial set of attribute values was elicited with the question "What do you correct in an FDEF to fix a defect?". This question provided a clear focus of discussion and greatly clarified the meaning of the classification scheme. The measurement team recorded the answers in a template as shown in Figure 9.

This template shows that definitions describing the fix clarify an attribute value and example defects are also shown. This is to support the clarity of the scheme and contribute to its reliability.

Attribute FDEF-Correction Type: What had to be corrected?		
Value	**Definition**	**Example**
Initialization	The initialization of a variable/routine/class has to be added or changed	Class behavior wrong in first call
…	…	…

Figure 9: Excerpt from Attribute Definition

These templates were sent to the interviewees for verification and completeness (e.g., giving example defects for each attribute value). In a second step, the developed scheme was presented to a broader audience in a workshop and checked for completeness, understandability, orthogonality, and meaningfulness.

The feedback from the workshop was clearly positive. The participants clearly regarded their involvement as positive, preferring to having a scheme imposed from the outside. Thus, the involvement of developers led to better acceptance of the scheme. Moreover, the initial development of a scheme in interviews was seen as positive, as in a small group creative discussions are possible, whereas the larger group has a concrete result to give feedback for.

7 Validation of the Classification Scheme

Currently we are performing a pilot project in which we introduce the customized defect classification scheme into the development process, collect data, and perform initial quantitative and qualitative analyses with respect to the quality of the scheme. The quality of a classification scheme is determined by two main factors: the reliability of the scheme and the ability to perform useful analyses.

The reliability of a classification scheme is determined by the question of whether different developers classify the same defects with a high degree of agreement. The main objective of our customization approach was just this: By defining the attribute values based on domain knowledge, giving them semantics, and enhancing the definitions with company-specific examples, we expect high reliability of the data. Nevertheless, we need to investigate this in more detail.

In order to investigate the usefulness of the classification scheme and whether meaningful analyses can be performed, it is important to consider several sub-aspects. In order to draw reasonable conclusions from the data it is essential that the scheme provides a stable defect distribution when the development process is stable. Without stability, it is impossible to decide whether

changes in the distribution are caused by chance or due to introduced process changes. In consequence, the second important aspect to consider with respect to the usefulness of the scheme is whether it is possible to monitor changes in the process by means of changes in the defect distribution. Finally, the usefulness is also determined by the question whether it is possible to identify improvement potential based on these distributions.

Reliability

In order to assess the scheme's reliability we followed the approach reported in [10]. The idea of this approach is that two different developers classify the same set of defects. The classification scheme has a good reliability if different developers select the same classes for the defects. Thus, we extracted a sample of 68 FDEF defects from the defect database, which were already classified by the developers fixing the defect. The person responsible for the *FDEF correction type* re-classified the defects according to his understanding of the scheme. To judge the reliability of the classification scheme, we compared the two classifications of the defects using Cohens Kappa [10]. To do so, the data were arranged as shown in Table 3. The table contains the proportion of classifications that fall in each of the cells.

		Recorder 1				
		$Class_1$	$Class_2$	…	$Class_n$	
Recorder 2	$Class_1$	P_{11}			P_{1n}	P_{1+}
	$Class_2$	P_{21}			P_{2n}	P_{1+}
	…					
	$Class_n$	P_{n1}			P_{kk}	P_{n+}
		P_{+1}	P_{+2}	…	P_{+n}	

Table 3: k x k Table for Cohens Kappa

Cohens Kappa κ is defined as:

$$\kappa = \frac{P_0 - P_e}{1 - P_e} \text{ with } P_0 = \sum_{i=1}^{k} P_{ii} \text{ and } P_e = \sum_{i=1}^{k} P_{+i} \times P_{i+}$$

In the current study we obtained a κ of 0.65. According to [10] values above 0.6 indicate good agreement (i.e., a good match of the classification) and values above 0.75 indicate excellent agreement. Thus, our scheme can be used with "good agreement" between individual developers. However, [10] indicates improvement potential, since they report that for code defect classifications, a κ of 0.88 can be achieved.

In order to identify those attribute values that cause the most disagreement, we determined the proportion of disagreement for each attribute value *j*

$$p_{disagr}(j) = \sum_{i=1}^{k} P_{ij} + \sum_{i=1}^{k} P_{ji} - 2 \times P_{jj}$$

In our study we identified three attribute values that were particularly prone to disagreement. For example, the two developers often disagreed on logic and documentation defects. Using the results of our study, we are now able to improve the definitions and training for these attribute values, which should result in improved reliability.

Stability of Distribution for Unchanged Process

To validate the stability of the defect distribution, we investigate defect distribution from the function review. We divided the set of function reviews into two groups of the same size, determined the *correction type* distribution, and analyzed differences in the distribution. The result is shown in the left panel of Figure 10 (for confidentiality reasons we removed the attribute values names). It can be observed that for 9 attribute values, the difference amounts 3% or less. Only one attribute is outlying with a difference of 8%. Based upon this result we conclude that the type distribution in function reviews is rather stable. Thus, the impact of future process changes should be observable in the type distribution. However, the outlier indicates that in interpreting two different distributions (e.g., before and after a process change), the variance in these data needs to be taken into account. Consequently, the type distribution needs to be monitored in order to determine its natural variation.

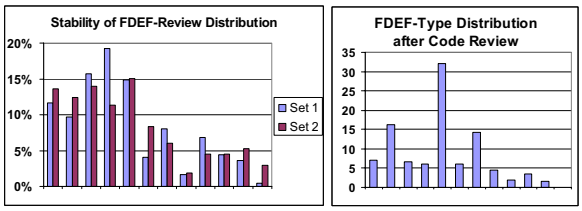

Figure 10: Defect Data from Case Study

Improvement Potential

The right panel of Figure 10 shows the defect distribution for FDEF defects detected after code review. The figure shows three correction types that have high proportions. These defect types frequently escape the review processes, since they are identified in later activities. Thus, these defect types provide a good potential for improving function reviews. These defect classes were a first input for discussions with process experts at Bosch GS. They confirmed that the identified classes should be addressed in earlier quality assurance activities. As a result of these observations, a change in the FDEF review process was triggered. Therefore, we conclude that the scheme also fulfills the objective of identifying improvement potentials in the development process.

Qualitative observations

In order to judge the usefulness of the scheme and its analyses, we interviewed representatives from the developers and the quality assurance (QA) team.

The developers considered the scheme as beneficial and useful for identifying those types of defects that slip through quality gates (e.g., inspections). This knowledge is seen as an important step to improve these activities.

According to the QA team, the scheme and its measurement concept have already established as an integral aspect of the company's quality assurance strategy, which emphasizes quantitative management and control of the quality assurance process in the context of CMMi Level 4. The ability to generate quantitative evidence about the status of the quality assurance process and the impact of process spanning changes, and the ability to present this evidence to management, was seen as beneficial. Finally, the QA team regarded the intensive involvement of developers and management in the framework of our process as helpful for obtaining support and sponsorship across all hierarchy levels.

8 Summary and Outlook

In this paper we presented an approach for defining, introducing, and validating customized defect classification schemes in industrial settings. In order to combine measurement know-how and domain know-how, great emphasis is put on performing structured interviews with software developers. The involvement of the developers provides good acceptance of the scheme and the proposed analyses are regarded as a beneficial measurement instruments for the systematic integration of various quality assurance processes.

Our experience in implementing defect classification at Bosch GS using the proposed approach is clearly positive. Due to the results we obtained, an extension of the measurement concept to other parts of the entire development process (e.g., activities before function specification) is already planned.

Acknowledgements

We thank all interviewees for their valuable contributions. In particular, we thank Mr. Müller, Ms. Rehner, and especially Mr. Walz and Ms. Weller for their support. We also thank Sonnhild Namingha for proofreading the paper.

References

[1] B. Anda,; D. Sjøberg; Towards an inspection technique for UC models; Proc. of the 14th Int. Conf. on Software Engineering and Knowledge Engineering, p. 127-134, 2002

[2] K. A. Bassin, T. Kratschmer, and P. Santhanam. Evaluating Software Development Objectively. IEEE Software, 15(6):66-74, Nov. 1998.

[3] B. Boehm,V.R. Basili. Software Defect Reduction Top 10 List, IEEE Computer, vol.34 no1, pp.135--137, 2001.

[4] L. C. Briand, B. Freimut, and F. Vollei, Assessing the Cost-Effectiveness of Inspections by Combining Project Data and Expert Opinion, Proc. of the 11th Int. Symp. on Software Reliability Engineering, pp. 124-135, 2000.

[5] D. Card, Managing Software Quality with Defects, Crosstalk - The Journal of Defense Software Engineering, March 2003.

[6] J. K. Chaar, M. J. Halliday, I. S. Bhandari, and R. Chillarege. In-Process Evaluation for Software Inspection and Test. IEEE Transactions on Software Engineering, 19(11):1055-1070, Nov. 1993.

[7] Chillarege, R; Bhandari, I; Chaar, J; Halliday, M; Moebus, D; Ray, B; Wong, M; Orthogonal defect classification -- A concept for in-process measurements, IEEE Transactions on Software Engineering, vol. 18, pp. 943-956, Nov. 1992.

[8] Denger, C.; Paech, B.; Freimut B., Achieving High Quality of Use-Case-Based Requirements, Informatik- Forschung und Entwicklung (accepted for publication)

[9] C. Ebert, R. Dumke, R. Bundschuh, Best Practices in Software Measurement, Springer, 2004.

[10] K. El Eman and I. Wieczorek, The Repeatability of Code Defect Classification, Proc. of the Int. Symp. on Software Reliability Engineering, 1998.

[11] B. Freimut, Developing and Using Defect Classification Schemes, Technical Report, IESE Report No. 072.01/E, 2001.

[12] B. Freimut, B. Klein, O. Laitenberger, G. Ruhe, Measurable Software Quality Improvement through Innovative Software Inspection Technologies at Allianz Life Assurance, Proc. of the ESCOM, pp. 345-353, 2000.

[13] R. Grady, Practical Software Metrics for Project Management and Process Improvement. Prentice Hall, 1992.

[14] M. Halliday, I. Bhandari, J. Chaar, R. Chillarege, Experiences in Transferring a Software Process Improvement Methodology to Production Laboratories, Journal of Systems and Software, vol. 26, pp. 61-68, 1994.

[15] W. Humphrey, A Discipline for Software Engineering, Addison-Wesley, 1995

[16] O. Laitenberger, S. Vegas, M. Ciolkowski, The State of the Practice of Review and Inspection Technologies in Germany, ViSEK Technical Report, ViSEK/011/E, 2002.

[17] T. Messmore, Process Metamorphosis in IBM's Storage System Division, Proc. of the Int. Conf. on Applications of Software Measurement, 1999.

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

[19] N. Nagappan, L. Williams, J. Hudepohl, W.Snipes, M. Vouk, Preliminary Results on Using Static Analysis Tools for Software Inspection, Proc. of the 15th Int. Symp. on Software Reliability Engineering, pp. 429-439, 2004.

[20] T. J. Ostrand and E. J. Weyuker, Collecting and Categorizing Software Error Data in an Industrial Environment, Journal of Systems and Software, vol. 4, pp. 289-300, 1984.

[21] A. Porter, L. G. Votta, V. R. Basili, Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment, IEEE Transactions on Software Engineering, 1996, pp 563-575.

[22] C. P. Schultz. Orthogonal Defect Classification (ODC)-based Test Planning and Development. Proc. of the Int. Conference on Applications of Software Measurement, 1999.

[23] R. van Solingen, E. Berghout, The Goal-Question-Metric Method, McGraww-Hill, 1999

[24] D. Vose, Quantitative Risk Analysis: A Guide to Monte Carlo Simulation Modeling. John Wiley Sons, 1996