Why Do We Not Learn from Defects?
Towards Defect-Driven Software Process Improvement

Niklas Mellegård[1], Miroslaw Staron[1] and Fredrik Törner[2]
[1]*Dept. of Computer Science, Chalmers University of Technology & University of Gothenburg, Gothenburg, Sweden*
[2]*Volvo Car Corporation, Gothenburg, Sweden*

Keywords: Software Engineering, Software Quality, Metrics/Measurement, Defect Classification.

Abstract: In this paper, we put forth the thesis that state-of-the-art defect classification schemes – such as ODC and IEEE Std. 1044 – have failed to meet their target; limited industrial adoption is taken as part of the evidence combined with published studies on model driven software development. Notwithstanding, a number of publications show that defect reports can provide valuable information about common, important, or dangerous problems with software products. In this paper, we present the synthesis of two industrial case studies that illustrate that even expert judgement can be deceptive; demonstrating the need for more objective evidence to allow project stakeholder to make informed decisions, and that defect classification is one effective means to that end. Finally, we propose a roadmap that will contribute to improving the defect classification approach, which in consequence will lead to a wider industrial adoption.

1 INTRODUCTION

Software defect classification schemes – such as ODC, HP and IEEE 1044 – have the purpose of providing defect reports with a common structure. Such a structure allows for efficient quantitative analyses, which can provide evidence of the efficiency and effectiveness of various process activities. Following ODC, defect classification schemes (DCS) have been around for more than two decades – during which, software development has evolved from being code- and document centric to be model-driven. Based on the number of publication in the area, however, we conjecture that DCS have had limited industrial impact – this limited impact is taken as a symptom of that the approach has failed to meet its target.

Despite limited adoption, publications – our own case study included – show that defect reports can provide valuable information for improving modelling when aggregated and analysed; to be an efficient tool to draw attention of various stakeholders to the most common, important or dangerous problems with software products.

We approach this apparent contradiction by addressing the question: "*As academic evidence show that DCS can be successful, why has it not had a more industrial impact and what can be done?*" We first address this question by concretely showing the value of DCS, using the synthesis of two industrial case studies from our previous work. We then provide evidence in support of the conjectured limited industrial adoption of DCS, and present reasons why. Finally, we provide a roadmap that would fill the gaps in current state-of-research that we envision would allow for a more successful approach to DCS.

In particular, we envision that the roadmap will contribute to making the results of defect analyses more useful to project stakeholders in control of resources, in particular in the system modelling phase. This is in contrast to the current state-of-the-art where analyses of classification data primary are intended for developers. By refocusing the DCS approach we envision that it will better serve as a tool for fact-based decisions during modelling – based on descriptive and predictive measures and indicators. The improvement would contribute to more accurate targeting of process improvement initiatives, to serve as the basis for defect-driven software improvement initiatives.

In practice, the purpose of a defect report is often limited to facilitating the resolution of the defect. For instance, defect reports are often free text (Wagner, 2008) which makes quantitative analyses effort intensive. In response, various DCS have been

Mellegård N., Staron M. and Törner F..
Why Do We Not Learn from Defects? - Towards Defect-Driven Software Process Improvement.
DOI: 10.5220/0004345602970303
In *Proceedings of the 1st International Conference on Model-Driven Engineering and Software Development* (MODELSWARD-2013), pages 297-303
ISBN: 978-989-8565-42-6
Copyright © 2013 SCITEPRESS (Science and Technology Publications, Lda.)

proposed. DCS also contribute to comparability of defect metrics between projects and between companies (Chillarege et al., 1992).

Classification schemes typically define a set of attributes, where each attribute captures a specific aspect of the defect – e.g. how the defect was detected, its severity and type. Each attribute typically contain a set of values that can be chosen from; this contributes to the efficiency and reliability of the classification. In literature, the most commonly referred (Freimut, 2001) DCS are ODC (Orthogonal Defect Classification) (Chillarege et al., 1992), the HP approach (Grady, 1992) and the IEEE Std. 1044 (IEEE, 2010).

The attributes of ODC and IEEE Std. 1044 are organized into the defect's life-cycle phases; ODC, for instance, defines the phases open and close (shown in Table 1). The attributes in the opener section of ODC focus on aspects of the failure, whereas the closer section focuses on aspects of the fault.

Table 1: Overview of ODC (adapted from (Freimut, 2001)).

Process	Attribute	Meaning
Open	Activity	When did you detect the defect?
	Trigger	How did you detect the defect?
	Impact	What would the customer have noticed if the defect had escaped into the field?
Close	Target	What high level entity was fixed?
	Source	Who developed the target?
	Age	What is the history of the target?
	Type	What had to be fixed?
	Qualifier	Was the defect caused by something missing, incorrect or extraneous?

DCS typically focus on technical aspects of the defects and their source code manifestations; IEEE Std. 1044, for instance, lists 80 different values for its *Type* attribute.

2 IMPORTANCE OF DEFECT CLASSIFICATION

In our earlier work (Mellegård and Staron, 2010) we investigated the importance of various artefact types in the automotive software development – such as requirements, types of software models, and documents. Specifically, we investigated the perceived importance of the artefacts and the relative effort required to create them. The particular focus of the case study was to characterize the use of software models in relation to other types of development artefacts.

Among the conclusions of the case study were that most effort was spent on simulation models (e.g. Simulink models), while the most important artefacts were the requirements and design artefacts. This result was in itself not surprising as the simulation models serve as a base for the implementation, and as development is highly distributed – both among in-house teams and external suppliers – the quality of specifications is crucial to preventing eventual instegration problems. In fact, during the case study we repeatedly encountered statements from our interviewees – expert opinions – that integration was a considerable challenge, in particular during the late project phases. Additionally, we frequently encountered concerns about lack of more objective evidence to support these expert opinions.

These findings directed our interest towards in-process defects, specifically to defects detected during late project phases; did integration issues cause these, and could we find evidence that the cause was as had been anticipated?

2.1 Cause of Late Defects

In a second study (Mellegård et al., 2012a), we set out to make an in-depth examination of in-process defects from one system developed at the department. We found, in the initial analyses of the existing defect data, that there was indeed a substantial inflow of defects in late project phases; shown in Figure 1 as the defect backlog.

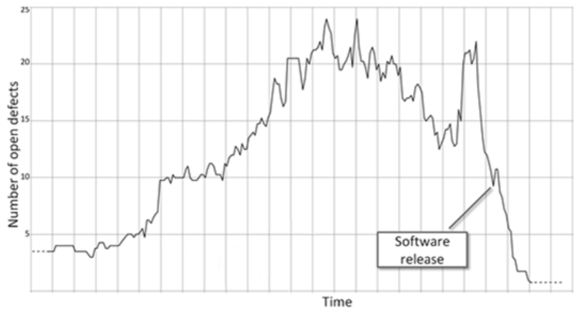

Figure 1: Defect backlog.

The timing of the late spike in defects close to software release (a major in-development milestone) seemed to confirm the hypothesis of integration issues. Merely examining the quantity of defects, however, would not reveal the nature of the defects and as the defect reports were in free text, they were not suitable for quantitative analysis. We therefore used IEEE Std. 1044 as base to develop a DCS adapted to the context of our case company. The result was the Light-Weight Defect Classification scheme (LiDeC) (Mellegård et al., 2012b).

As part of the case study, LiDeC was applied to a sample of defects from the late defect inflow spike as shown in Figure 1 and reported in (Mellegård et al., 2012a), and the analysis results were presented and discussed at a workshop at the company.

Figure 2 shows a sample of the analysis using four attributes from LiDeC (see (Mellegård et al., 2012b), from three perspectives: detection of the failure, type of fault and finally product and project impact of the resolution. Using these three perspectives, the defects can be examined to evaluate whether they were integration issues as anticipated.

Figure 2 shows that while the majority of defects was indeed detected during integration testing – system or functional – the defect types were not typical for integration issues. Whereas the anticipated type would be *Interface, Data* or *Tuning*, the majority of defects were of the type *Logic* –i.e. computational or algorithmic faults. Such defects would normally be present already in the simulation models. This was corroborated by the *Resolution impact* attribute, shown in the bottom left of Figure 2, indicating that most defects required changes to a single unit –integration issues would typically have an impact on multiple units, or require changes to the specification (denoted as *Functional changes*).

However, the *Re-verification Level* attribute – showing the activity required to test a resolution – indicates that it is not a clear-cut case. On the one hand, a significant amount of defects requires only inspection or component test, indicating unit problems. On the other hand, most defects would require new system or functional tests, indeed indicating integration impact – finding the root cause of these defects could bring significant benefits.

Our (careful) conclusions from this study is that the majority of late defects – although to a large extent requiring new integration tests – are not of the type typically associated with integration problems. Thus, the classification of defects provided the development teams with new information that may contribute to better test planning – e.g. put effort into improving testability of requirements on unit level.

Finally, we would like to emphasize that the analysis presented here was conducted on a sample of defects from a project that had finished a year prior to the study. The results should therefore be treated as proof-of-concept rather than as a basis for recommended change of practice. We can however conclude that the classification contributed with new information that in part contradicted expert opinion. This raised interest and inspired discussions regarding possible causes; the case study provides evidence that conducting defect classification and analysis contributes to constructive review of the state-of-practice.

3 PROBLEM DESCRIPTION

As we illustrated above, the use of DCS can be an effective approach to extracting data from problem reports, and can provide new information about the development process. In addition, there have been studies reporting similar results, e.g. Butcher et al. (Butcher et al., 2002), or Li et al. (Li et al., 2012).

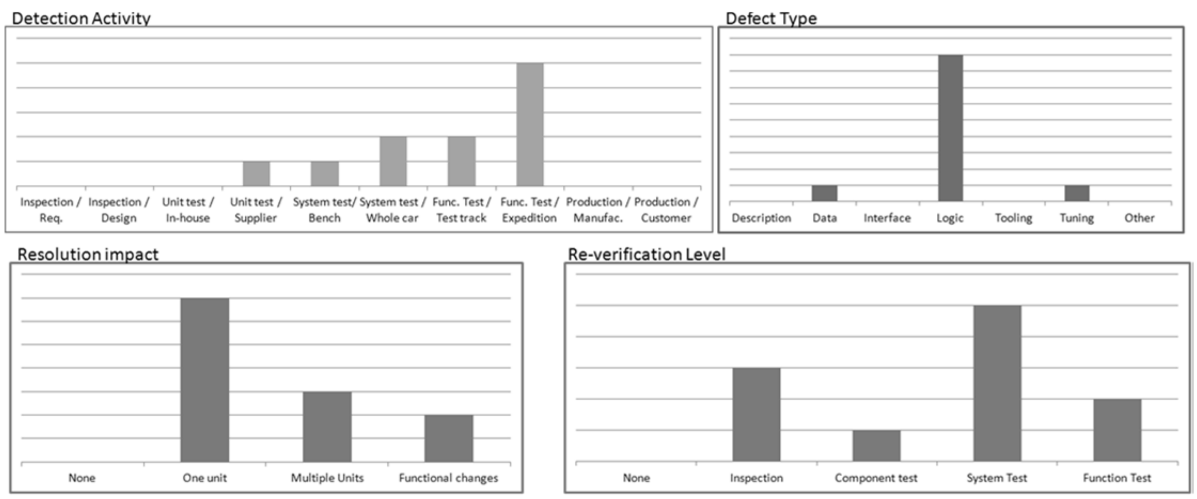

Figure 2: Preliminary analysis results.

However, we conjecture – partially based on our observations – that DCS has had limited industry adoption; we take this as a symptom that state-of-the-art DCS as a means of extracting process metrics has missed its target. To find evidence in support of this conjecture we searched IEEE Explore (http://ieeexplore.ieee.org) using the search term: ('defect classification' AND 'software'). The search, performed Oct. 25 2012, yielded 70 publications between 1986 and 2012. By reading titles and abstracts, we found that the publications fell into the following categories:

- *Proposing new DCS*, e.g. (Chillarege et al., 1992); (Paul et al., 2002); (IEEE, 2010)
- *Improving existing DCS*, e.g. by assisting a user in conducting the classification (Wang He et al., 2009); (Huang et al., 2011), or adapting the scheme to a specific context (Li et al., 2010)
- *Academic evaluation* of a DCS, e.g. (Henningsson et al., 2004); (Vetro' et al., 2012)
- *Industrial evaluation* of a DCS, e.g. (Butcher et al., 2002); (Chillarege et al., 2002); (Freimut et al., 2005); (Li et al., 2012)
- *Analysis techniques* for classification data (Li et al., 2010)
- *Using classification data* for other purposes, e.g. to evaluate efficiency of inspections (Nagappan et al., 2004), to evaluate static analysis for fault detection (Zheng et al., 2006), to propose reliability estimation models (Paul et al., 2000), or to evaluate fault injection techniques (Jin et al., 2009)

Notably, we found no publications evaluating industrial adoption of DCS, nor investigating what companies would require from such an approach; specifically in terms of the information that analyses of the data need to provide.

Further support for our conjecture can be found in the systematic literature review by Hall et al. (Hall et al., 2011). In their paper, Hall et al. examined 36 fault prediction models and noted that the vast majority of the models were limited to predicting the quantity of faults per module. In fact, Hall et al. could only find one model that incorporated fault severity as a predicted variable. In their paper, Hall et al. argue that one reason may lay in the difficulty of defining severity. An additional reason, however, may simply be a lack of available data; companies tend to collect only the defect data necessary to facilitate the resolution of the defect. This brings our thesis to a point: *although defect classification has been shown to be an effective approach to acquiring process metrics, why has it not had a wider industrial adoption?*

We can consider this point in the context of communication paths (Pareto et al., 2012). In their paper, Pareto et al. argue that a source of problems in projects is miscommunication because the needs and concerns of developers are not expressed in terms that managers and architects need in order to make informed decisions – rather developers express their needs in highly detailed and specific technical terms. In order to make an impact on the stakeholders in power, developers need to create abstractions suitable for that specific stakeholder; to provide evidence that level of abstraction.

In this context, we contend that established DCS are too focused on the developers' context – illustrated, for instance, by the high granularity of the *Type* attribute in both ODC and the IEEE 1044. In particular, established DCS fail at providing sufficient guidance to translate the results into the language needed to make an impact on the stakeholders that are in control of the resources.

Furthermore, unnecessarily high level of detail in classification brings a risk that may have a double impact: on the one hand, it adds to the effort needed to make a classification (and thus reduces the available resources to resolve the problem). On the other hand, it adds to the required analysis effort needed to adapt results to the stakeholders in control. Consequently, effort risks being put into collecting data that remain unused (Li et al., 2012).

4 ROADMAP

The roadmap is divided into three parts: investigation of current DCS state-of-practice; investigation of how the design of DCS could meet the needs better; and finally, investigation of how to analyse the data to meet the organizational information needs.

4.1 State-of-practice

As we found a notable lack of research into current DCS state-of-practice in general and in modelling specifically, we envision surveying:

- To what extent do companies use DCS – in particular in the context of how DCS provides input to decision formulation and execution processes;
- What alternative approaches are used to facilitate analyses of defect reports on an aggregate level (e.g. none, defect taxonomies, root-cause analysis etc).

Furthermore, the results of the surveys should be correlated with aspects such as the size of the company and development teams, types of products developed and types of development processes used. Such an in-depth understanding of current practice would contribute to improving state-of-the-art DCS.

Additionally, there is a need to investigate the information needs of relevant project stakeholders – mainly product and project managers, and architects. In their paper Buse and Zimmerman (Buse et al., 2011) examine general information needs among the stakeholders at a large software organization, exemplified by Microsoft. We, however, call for a more targeted investigation into which stakeholders are relevant, and what their information needs are, with the specific focus on defect data. We envision the needs falling into two main categories: descriptive and predictive.

Descriptive information would characterize project phases in terms of their defect profile (patterns). Descriptive information could be used in-process for benchmarking against a company base-line (Chillarege et al., 1992), for instance to provide evidence for evaluating process improvements.

Predictive information needs relate to the challenges of resource planning. For instance, in assigning resources of test phases in a project, there may be a need to predict the anticipated amount and type of defects – fault prediction models, such as reviewed in Hall et al. (Hall et al., 2011) aim at that. However, more granular defect data may enable more precise prediction models, thus enable defect-driven proactive decision support.

4.2 Design of DCS

The state-of-practice investigations should be complemented with establishing a library of best practices and lessons learned in both the design and application of DCS. Li et al. (Li et al., 2012) made a recent contribution in part by reporting a number of lessons learned when applying DCS in two organizations, however more studies are necessary.

Even though the design of DCS is already well represented in state-of-the-art, there are aspects that are not sufficiently developed. For instance, DCS should be refocused from aspects of the implementation (source code) to covering all project phases – in particular modelling. There is, additionally, a need to build abstraction mechanisms into the DCS in order to reduce the required analysis effort, and to improve the comparability of data between projects and organizations. LiDeC (Mellegård et al., 2012a, 2012b) contributes to this for the automotive domain, but studies in other domains are needed. For instance, the attribute values in LiDeC are structured hierarchically, which is an inherent abstraction mechanism. This allows attributes to be extended with values at more detailed levels while retaining comparability at higher levels of abstraction.

In addition, the design of DCS should maintain reference to the ISO/IEC 15939 standard (ISO/IEC, 2007) in order to facilitate integration with other measures (e.g. predictions).

4.3 Data Analysis

The arguably most challenging aspect of DCS is in analysing of the data. The thesis put forward in this paper is that state-of-the-art DCS have failed partly due to insufficient analysis methods. We propose therefore the need for research into analysis and visualization methods that satisfy typical information needs and attract attention to the most important defect patterns (as proposed in section 4.1). For instance, identifying product and project characteristics, such as change patterns in source code or software models (by inspecting versioning systems), that correlate with defect inflow profiles would enable defect inflow prediction models based on data mined from software repositories.

We envision an analysis reference manual that maps a stakeholder's information need with a set of best practices – for instance as a recommendation on which attributes to include in the analysis and how to visualize the data.

Moreover, we assert that reporting on industrial case studies where specific organizational problems have been addressed by analysis of defect classification data would be of valuable – the work by Li et al. (Li et al., 2012) contributes to this end.

5 CONCLUSIONS

In this paper, we have examined defect classification schemes as a tool for collecting process metrics in model based automotive software development projects. Specifically, we have critically examined the quality of state-of-the-art defect classification by investigating its industrial adoption. Our thesis was that defect classification has had limited industrial adoption which we have argued to be a symptom of knowledge gaps in state-of-the-art DCS.

One main reason for limited industrial adoption is – in our view – that state-of-the-art DCS are inadequate for their purpose. In particular, there is a too strong of a focus on low-level aspects of the implementation; i.e., a tool primarily intended for developers. DCS thus fail to address that project stakeholders in control of resources need information on a different level of abstraction to make informed decisions. This means that state-of-the-art classification approaches are poorly designed to produce the results that are needed in order to make an impact in an organization; thus the effort invested in collecting data risks being in vain, as a large potential of the data remain unused.

We have proposed a roadmap for an improved defect classification approach that would contribute towards developing new proactive evidence-based software process improvement strategies – defect-driven software process improvement. The roadmap includes: making a deeper investigation of the current adoption rate in industry; investigation of the typical information needs of the project stakeholders that have control over resources; investigation of how to design DCS to support multiple levels of abstraction, and finally; to investigate methods of data analyses to accommodate the information needs of the various project stakeholders.

These actions will contribute to making DCS more appropriately adapted to organizations' needs. This in turn, we conjecture, will result in wider industrial adoption.

ACKNOWLEDGEMENTS

This research is partially sponsored by The Swedish Governmental Agency for Innovative Systems (VINNOVA) under the Intelligent Vehicle Safety Systems (IVSS) programme.

REFERENCES

Buse, R. P. L., Thomas Zimmermann, 2011. *Information Needs for Software Development Analytics* (Microsoft Technical report No. MSR-TR-2011-8)

Butcher, M., Munro, H., Kratschmer, T., 2002. Improving software testing via ODC: Three case studies. *IBM Systems Journal 41*, 31 –44.

Chillarege, R., Bhandari, I. S., Chaar, J. K., Halliday, M. J., Moebus, D. S., Ray, B. K., Wong, M.-Y., 1992. Orthogonal defect classification-a concept for in-process measurements. *IEEE Trans. Softw. Eng. 18.*

Chillarege, R., Ram Prasad, K., 2002. Test and development process retrospective - a case study using ODC triggers, *Int'l Conf Dependable Syst. and Netw.*

Freimut, B., 2001. Developing and using defect classification schemes (IESE- Report No. 072.01/E). *Fraunhofer IESE.*

Freimut, B., Denger, C., Ketterer, M., 2005. An industrial case study of implementing and validating defect classification for process improvement and quality management, 1*1th IEEE Int'l Symp. Softw.* Metrics.

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

Hall, T., Beecham, S., Bowes, D., Gray, D., Counsell, S., 2011. A Systematic Review of Fault Prediction Performance in Software Engineering. *IEEE Trans. Softw. Eng.*

Henningsson, K., Wohlin, C., 2004. Assuring fault classification agreement - an empirical evaluation, *Int'l Symposium on Empirical Softw.* Eng. (ISESE).

Huang, L., Ng, V., Persing, I., Geng, R., Bai, X., Tian, J., 2011. AutoODC: Automated generation of Orthogonal Defect Classifications, 2*6th IEEE/ACM Int'l Conf. on Automated Softw.* Eng. (ASE).

IEEE, 2010. *IEEE Std. 1044-2009.* Standard Classification for Software Anomalies.

ISO/IEC, 2007. *ISO/IEC 15939* - Systems and Software Engineering – Measurement Process.

Jin, A., Jiang, J., 2009. Fault Injection Scheme for Embedded Systems at Machine Code Level and Verification, *15th IEEE Pacific Rim Int'l Symp. on Dependable Comput. (PRDC).*

Li, J., Stalhane, T., Conradi, R., Kristiansen, J. M. W., 2012. Enhancing Defect Tracking Systems to Facilitate Software Quality Improvement. *IEEE Softw. 29, 59 –66.*

Li, N., Li, Z., Sun, X., 2010. Classification of Software Defect Detected by Black-Box Testing: An Empirical Study, 2nd *World Congress Softw. Eng. (WCSE).*

Li, N., Li, Z., Zhang, L., 2010. Mining Frequent Patterns from Software Defect Repositories for Black-Box Testing, 2nd *Int'l Worksh. Intell. Syst. and Appl. (ISA).*

Mellegård, N., Staron, M., 2010. Characterizing Model Usage in Embedded Software Engineering: A Case Study, 8th *Nordic Workshop on Model Driven Softw.* Eng. (NW-MoDE). Copenhagen, Denmark.

Mellegård, N., Staron, M., Törner, F., 2012a. A Light-weight Defect Classification Scheme for Embedded Automotive Software and its Initial Evaluation, *IEEE Int'l Symp. Softw. Rel. Eng. (ISSRE)*, Dallas Tx USA.

Mellegård, N., Staron, M., Törner, F., 2012b. A Light-Weight Defect Classification Scheme for Embedded Automotive Software (Technical Report No. 2012:0x, ISSN: 1654-4870), Research Reports in Software Eng. and Management. Chalmers Univ. of Tech., Göteborg.

Nagappan, N., Williams, L., Hudepohl, J., Snipes, W., Vouk, M., 2004. Preliminary results on using static analysis tools for software inspection, 15th *Int'l Symp. Softw. Rel. Eng. (ISSRE).*

Pareto, L., Sandberg, A. B., Eriksson, P., Ehnebom, S., 2012. Collaborative prioritization of architectural concerns. *Journal of Syst. and Softw. 85.*

Paul, R. A., Bastani, F., I-Ling Yen, Challagulla, V. U. 2000. Defect-based reliability analysis for mission-critical software, *Comp. SW and Applications Conf. (COMPSAC)*.

Paul, R. A., Bastani, F. B., Challagulla, V. U. B., Yen, I.-L., 2002. Software measurement data analysis using memory-based reasoning, *14th IEEE Int'l Conf. Tools with AI, (ICTAI)*.

Wagner, S., 2008. Defect classification and defect types revisited, *Workshop on Defects in Large Softw. Syst., DEFECTS.*

Wang He, Wang Hao, Lin Zhiqing, 2009. Improving Classification Efficiency of Orthogonal Defect Classification via a Bayesian Network Approach, Comp. *Intell. and Softw. Eng. (CiSE)*.

Vetro', A., Zazworka, N., Seaman, C., Shull, F., 2012. Using the ISO/IEC 9126 product quality model to classify defects: A controlled experiment, *16th Int'l Conf. Eval. Assessment in Softw. Eng. (EASE)*.

Zheng, J., Williams, L., Nagappan, N., Snipes, W., Hudepohl, J. P., Vouk, M. A., 2006. On the value of static analysis for fault detection in software. *IEEE Trans. Softw. Eng.*