The effects of UML modeling on the quality of software

Nugroho, A.

Chapter 6

Level of Detail in UML Models and its Relation with Defect Density

Level of detail (LoD) is one aspect in modeling that is often applied diversely. It represents the amount of information that is used to specify models. However, the importance of LoD in modeling and how it might affect the implementation has not been studied. Therefore, in this chapter we propose some measures to quantify LoD. The proposed measures are applicable to UML class- and sequence diagrams, and are evaluated using a significant industrial Java system. Based on the case study we have found that LoD of messages in sequence diagrams is significantly correlated with defect density in the implementation.

6.1 Introduction

Understanding the impact of software models on downstream software development is important for delivering good quality software. This knowledge may help avert software problems in an early stage of development and thus reducing the costs for solving defects—the reason: fixing defects earlier is believed to be much cheaper than later during software development [17]. Therefore, in the quest for identifying early indicators of software quality, researchers have focused on the attributes of software designs and their relations to the quality of the final software product. To name just a few, the seminal paper of Chidamber and Kemerer was aimed at identifying useful metrics for object-oriented designs [35]. Assessments of design

This chapter is an extension of the paper entitled "Empirical Analysis of the Relation between Level of Detail in UML Models and Defect Density", published in the proceedings of the 11th International Conference on Model Driven Engineering Languages and Systems (MODELS) 2008.

metrics and their implications on software defects have been reported in [30, 57, 121]. Many more works are targeted at predicting fault-proneness of classes based on design metrics [15, 44, 68, 122].

While many previous works have confirmed the usefulness of several design metrics as quality indicators of software quality, the practicality of the proposed metrics for *early* quality indicators is not clear. To the best of our knowledge and experience, software models in industry (e.g., represented using UML) are rarely complete—that is, many model elements are not specified due to various reasons ranging from designer's style/preference to resource constraints considerations. As a result, measuring product metrics such as CBO (coupling between objects) from a class diagram of a UML model, for example, might give a totally different picture from the implementation [127].

It is precisely due to the above issue that we feel challenged to come up with design metrics that can be collected during the design phase of software development. The metrics that we propose are level of detail (LoD) metrics, which essentially measure the amount of information that is used to specify software models. In this chapter, we focus on UML as a modeling language, and the metrics proposed are applicable to class- and sequence-diagrams of UML models.

The main motivation of using LoD as a measure is that LoD in UML models varies widely across model elements, diagrams, and projects. Therefore, it is interesting, and yet crucial, to investigate whether the amount of information that is used to represent models has any correlation with the quality of the final implementation.

Using the GQM template [130], the objective of this study can be formulated as follows:

Analyze level of detail in UML models
for the purpose of investigating its relation
with respect to the quality of the implementation
from the perspective of the researcher
in the context of an industrial Java system

This chapter is organized as follows. In Section 6.2, we discuss related work. In Section 6.3, the notion of level of detail (LoD) and the proposed LoD measures are presented. Section 6.4 discusses the design of the study, and in Section 6.5 and 6.6, the case study and the results are discussed respectively. In Section 6.7, we discuss the interpretations of the results and limitations of this study. Finally, in Section 6.8 we draw some conclusions and outline future work.

6.2 Related Work

Many studies that investigated the usefulness of design metrics as indicators of software quality used the object-oriented metrics proposed by Chidamber and Kemerer (CK metrics) [35]. These studies generally investigated the relationships between design metrics and software quality attributes such as fault-proneness, productivity, and maintainability.

One of the early studies that investigated the relation between object-oriented metrics and software quality is from Li and Henry [81]. The authors assessed the impact of CK metrics on the number of changes in classes of two commercial systems implemented using an object-oriented dialect of Ada. The results of their study showed that CK metrics seemed to be reasonable predictors for the amount of changes in a class during maintenance.

Previous studies also explored subsets of CK metrics and their relations to software maintainability. The work of Harrison et al. [61] for example, reported a controlled experiment about the impact of inheritance levels on understandability and modifiability of object-oriented systems. Results obtained from the experiment suggested that systems without inheritance hierarchy were easier to modify than the corresponding system with three or five levels inheritance hierarchy. An earlier work by Briand et al. experimentally investigated the effect of applying design principles proposed by Coad and Yourdon [39] on the maintainability of an object-oriented design. The results showed that adherence to the design principles, which includes coupling, cohesion, and inheritance principles, improved the understandability and modifiability of the object-oriented design [25].

There have been many studies that explored the relations between object-oriented metrics and module fault-proneness. The work of Basili et al. for example, assessed the usefulness of CK metrics as predictors of class fault-proneness [15]. The authors found that five out of six CK metrics (i.e., WMC, DIT, RFC, NOC, and CBO) were significant predictors of class fault-proneness. The work of Brito e-Abreu et al. defined several object-oriented metrics and evaluated their relations with the quality of Ada system [30]. Results showed that the metrics defined might have strong correlations with the quality of the system. Briand et al. assessed several coupling measures that were conceptually different from CK's coupling metric [22]. The authors suggested that the coupling metrics were reasonable measures of fault-proneness, and considered them complementary to CK metrics. In a different study, Briand also explored the relationships between design measures such coupling, cohesion, and inheritance, and the probability of classes to contain faults [28]. The authors found that many of the measures actually capture similar dimension, and it was possible to construct accurate prediction models using a subset of the design measures. Other studies in this line of research include that of Cartwright and Shepperd [31] and Briand et al. [29]. Cartwright and Shepperd found that C++ classes participated in inheritance hierarchies were three times more fault-prone than those classes not in inheritance hierarchy. Briand et al. explored the relationships between OO design measures (i.e., coupling, cohesion, and inheritance) and the fault-proneness of C++ classes. The results showed that the frequency of method invocations (import coupling) seemed to be the major factor that drives class fault-proneness.

One of the studies that explored the relations between CK metrics and productivity was conducted by Chidamber et al. The authors assessed the impact of CK metrics on productivity, rework effort, and design effort. One of the results suggested that high coupling and lack of cohesion corresponded to lower productivity, greater rework, and greater design effort [34].

The aforementioned studies focused on assessing the quality of software models from their structural quality properties, i.e., using object-oriented design metrics. Some studies have also been done to assess the relations between styles/formality in software models and

software quality. Generally, these studies considered understandability and maintainability as quality indicators of software. For example, the work of Briand et al. experimentally assessed the impact of using OCL (object constraint language) in UML models on defect detection, comprehension, and impact analysis of changes [27]. Although the overall benefits of using OCL on the aforementioned activities are significant, they have found that the benefits for the individual activities are modest.

A study of modeling style from Staron et al. looked into the effect of using stereotypes on model comprehension. The result suggests that UML stereotypes with graphical representation improve model comprehensibility [120]. Ricca et al. also found that stereotypes have a positive impact on diagram comprehension [111]. However, this finding was particularly true for inexperienced subjects—the impact was not statistically significant for experienced subjects. Genero et al. studied the influence of using stereotypes in UML sequence diagrams on comprehension [53]. While this study revealed no significant impact, it suggested that the use of stereotypes in sequence diagrams was favored to facilitate comprehension. Another study was conducted by Cruz-Lemus et al. to evaluate the effect of composite states on the understandability of state-chart diagrams [40]. The authors stated that the use of composite states, which allows the grouping of related states, improves *understandability efficiency* when reading state-chart diagrams. Nevertheless, subjects' experience with state-chart diagrams was considered as a prerequisite to gain the improved understandability.

Another work worth mentioning is from Genero et al., which explored measures that could be used to predict diagram maintainability [54]. Their study revealed that the number of associations and the maximum depth of inheritance (DIT) in class diagrams are good predictors of the time required to understand (understandability time) and modify (modifiability time) the diagrams. An experimental study from Arisholm et al. looked at the problem from a coarser grained view: the absence/presence of UML in software maintenance [10]. In the experiment, the authors used students as subject and employed lab-designed UML models. The results of their study confirmed that the use of UML for maintenance significantly reduces the time to make code changes in the system and increases functional correctness of the changes. However, the authors also stated that effort saving was not visible when the time required to change the UML diagrams was taken into consideration.

Note that we also performed a somewhat similar study to that of Arisholm et al., in which we investigated the effects of modeling or not modeling a class on the defect density of that class in the implementation [97]—this study was discussed in Chapter 4. Based on an industrial case study, we have found that classes that are modeled (either in class- or sequence diagrams) have a significantly lower defect density in the implementation than those that are not modeled.

Different from the aforementioned studies on design metrics, in this work we essentially consider rigor in UML modeling, i.e., the use of details, instead of structural design measures as an indicator of model quality. Our work is also different from most studies that looked into modeling style and rigor in that we propose measures to quantify rigor in modeling and assess their correlation with defect density in software system.

6.3 Level of Detail in UML Models

6.3.1 Definition

In UML modeling, the level of detail can be measured by quantifying the amount of information that is used to represent a modeling element. For example, the modeling element 'message in a sequence diagram' may be represented in any of the following manners, which use different amounts of information:

- an informal label,
- a label that represents a method name,
- a label that represents a method name plus the parameter list,
- a label that represents a method name plus the parameter list and parameter types.

In modeling UML class diagrams, many syntactical features are available to increase the level of detail: class attributes and operations, association names, association directionality, multiplicity and so forth. When the level of detail used in a UML model is low, it typically employs only a few syntactic features such as class-name and associations without specifying any further details or information about the class.

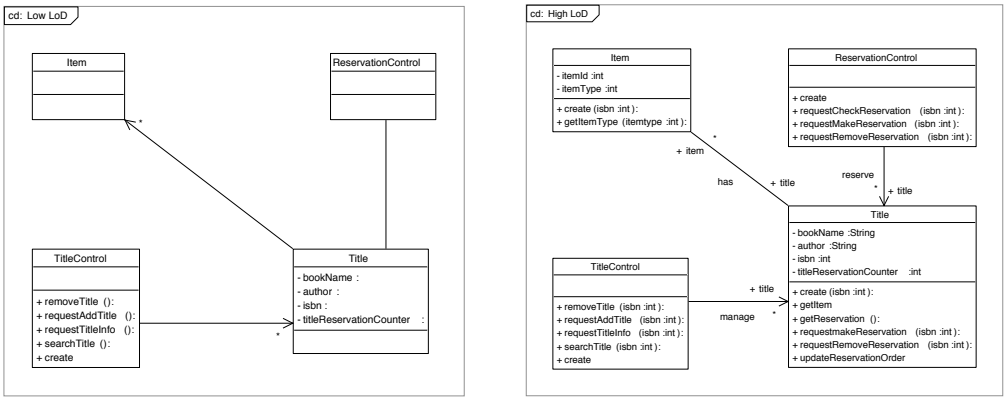
The way in which UML is used in general varies widely from project to project [79]. This variation includes the level of detail and completeness of the models. In this respect, the level of detail in UML models is important to assure that software developers can easily understand and implement them.

Recall the class diagram example (Figure 6.1) that was used earlier in Chapter 5. The figure shows an example of two structurally identical class diagrams which employ different level of detail. The class diagram with low LoD lacks information such as class attributes and operations. Even if attributes and operations are present, they might not be completely specified, e.g., in terms of attribute types or operation parameters. The same is true for associations, in which association names and role names are not specified. The class diagram with higher LoD provides more complete information on class properties.

6.3.2 Level of Detail Measures

The unit of analysis of this study is a class. Hence, the LoD measures defined are intended to quantify LoD at the level of individual classes.

Two sets of LoD metrics are defined to measure the LoD of classes modeled in UML models. One set measures LoD based on information in class diagrams, and the other set is based on information in sequence diagrams. The main reason of only using these two diagram types is that they are most commonly used in practice. This decision is also supported by survey findings reported in [42], which states that class, sequence, and use case diagrams are used in practice most often. Although use case diagrams are also commonly



(a) Low LoD

(b) High LoD

Figure 6.1: Identical class diagrams modeled with different LoD

used in practice, they merely describe the required functionality, but not how to implement this. Thus, except for missing requirements/functionality, other defect types are generally not related to use case diagrams.

Class Diagram LoD Metrics

We define four metrics to measure LoD from class diagrams. Compared to our previous study on LoD [98], we improve the practicality of the measures by excluding metrics that require information from source code. In this way, the new metrics can be better used for early prediction. The definitions of the class diagram metrics are provided in the following passages.

- **AttrSigRatio** ($CD_{attrsig}$) Measures the ratio of attributes with signature to the total number of attributes of a class.

  Attribute signatures indicate the types of data of class attributes. As such, attribute signatures provide important information for developers to implement class attributes or to manipulate data stored in an attribute.

- **OpsWithParamRatio** (CD_{opspar}) Measures the ratio of operations with parameters to the total number of operations of a class.

  Operation or method parameters denote inputs that are required by a method to perform its behavior. Although it is likely that some operations do not have parameters, especially those that do not deal with data manipulation, we believe that most operations require parameters—note that even trivial operations such as getters and setters require parameters.

- **AssocLabelRatio ($CD_{asclabel}$)** Measures the ratio of associations with label (i.e., association name) to the total number of associations of a class.

  Associations represent relationships amongst classes. Association names are important to clarify relationships amongst classes, in particular relationships that are not trivial or are not commonly known. As such, association names help readers to better understand concepts portrayed in a class diagram.

- **AssocRoleRatio ($CD_{ascrole}$)** Measures the ratio of associations with role name to the total number of associations attached to a class.

  When there is an association between two classes. A role name specified at one end of the association indicates the attribute name that must be implemented by the class at the other end of the association. As such, role names provide explicit specification on how to implement associations between classes.

As can be seen from the above list, all the metrics are measured in ratio. Ratio-based measures is preferred to conservative counting measures because the use of ratio measures neutralizes the dominant effects of size factors (e.g., the number of attributes or operations) on the LoD metrics.

We categorize the above metrics into two higher-level LoD measures, namely attributes and operations detailedness (CD_{aop}), and association detailedness (CD_{asc}). As such, for an implementation class x, and corresponding design class x' we define:

$$CD_{aop}(x) = CD_{attrsig}(x') + CD_{opspar}(x') \tag{6.1}$$

$$CD_{asc}(x) = CD_{asclabel}(x') + CD_{ascrole}(x') \tag{6.2}$$

As we can see in (6.1) and (6.2), the LoD measures of a class modeled in class diagrams are determined based on information that is related only to that particular class (measured at class level), rather than on the LoD of the class diagram as a whole in which a class appears (measured at diagram level). We need to distinguish this approach from the next because for sequence diagrams we measured LoD at diagram level.

Sequence Diagram LoD Metrics

The approach for measuring class LoD using sequence diagram metrics is different from that of using class diagram metrics. Firstly, the LoD of a class is determined based on the LoD of the sequence diagram—in which the class appears—in its entirety (i.e., not limited to the corresponding class object/instance in a given sequence diagram). Secondly, because a class may appear more than once in sequence diagrams, the LoD of a class is determined as the average LoD of the sequence diagrams in which it appears.

Five metrics are defined to measure LoD of a class based on its appearance in sequence diagrams. Two metrics that were introduced in the earlier study [98] are excluded, namely MsgWithLabelRatio and MsgWithGuardRatio. These metrics are excluded because after a reexamination they do not sufficiently capture level of detail in sequence diagrams. The metrics used in this study are described in the following passages.

- **NonAnonymObjRatio (SD_{nanobj})** Measures the ratio of objects with a name to the total number of objects in a sequence diagram.

  Objects in a sequence diagram represent instances of classes that interact to realize a given scenario or functionality. Providing each object with a unique name will reduce confusion when there are two objects of the same class being instantiated. Further, naming objects consistently will also ease implementation and improve traceability between models and the implementation.

- **NonDummyObjRatio ($SD_{ndumobj}$)** Measures the ratio of objects that correspond to classes that are modeled in class diagrams to the total number of objects in a sequence diagram.

  Specifying objects without proper references to the classifiers is another form of lacking of detail in describing objects in sequence diagrams. Similar to the previous metrics, fake object classifiers lead to problem related to traceability and more importantly, consistency.

- **NonDummyMsgRatio (SD_{ndumsg})** Measures the ratio of messages that correspond to methods specified in class diagrams to the total number of messages in a sequence diagram.

  This metric measures a similar aspect as the above two metrics, but focuses on messages. Messages without clear references to existing methods can cause confusions in the implementation, in particular if they are specified incompletely.

- **ReturnMsgWithLabelRatio (SD_{ret})** Measures the ratio of return messages with a label (any text attached to the return messages) to the total number of return messages in a sequence diagram.

  Labels in return messages indicate the type of data returned by message calls. A clear description of labels in return messages, e.g., important data that needs to be returned, helps developers implement message calls correctly.

- **MsgWithParamRatio (SD_{par})** Measures the ratio of messages with parameters to the total number of messages in a sequence diagram.

  Message parameters denote inputs required by an operation to perform its behavior (see also CD_{opspar}). Hence, incomplete specification of message parameters might cause implementation of message calls with a wrong parameter set, unmatched to the corresponding class method. Obviously, the problem is more severe if no corresponding method is specified.

As with the class diagram LoD metrics, the sequence diagram LoD metrics are also measured using ratios. In essence, the sequence diagram metrics cover two aspects of detailedness, namely *object detailedness* and *message detailedness*—SD_{nanobj} and $SD_{ndumobj}$ belong to object detailedness, and SD_{ndumsg}, SD_{ret}, and SD_{par} belong to message detailedness. Consequently, for a sequence diagram i we define measures of object detailedness and message detailedness as in formula (6.3) and (6.4) respectively:

$$ObjLoD = SD_{nanobj(i)} + SD_{ndumobj(i)} \tag{6.3}$$

$$MsgLoD = SD_{ndumsg(i)} + SD_{ret(i)} + SD_{par(i)} \tag{6.4}$$

To calculate the LoD of a class, we first define object detailedness and message detailedness of every sequence diagram in which a class appears using formula (6.3) and (6.4). Following this, because a class may appear in multiple sequence diagrams, the LoD of a class is defined by taking into account all sequence diagrams in which that particular class appears.

To normalize the effect of the number of occurrences of a class across multiple sequence diagrams on its LoD scores, the score of a given class is defined as the average of LoD scores (i.e., the score for $ObjLoD$ and $MsgLoD$) of the sequence diagrams in which it appears. Therefore, for an implementation class x, a corresponding design class x' and n sequence diagrams in which x' occurs, we define object detailedness (SD_{obj}) and message detailedness (SD_{msg}) as follows:

$$SD_{obj}(x) = \frac{1}{n} \sum_{x' \in n}^{n} ObjLoD(x') \tag{6.5}$$

$$SD_{msg}(x) = \frac{1}{n} \sum_{x' \in n}^{n} MsgLoD(x') \tag{6.6}$$

In addition to this approach of measuring LoD at diagram level, we considered measuring the LoD of a class at object level—that is, the LoD of an implementation class is measured from the detailedness of the corresponding object modeled in sequence diagrams. For example, NonDummyObjRatio would then be defined as the ratio of the number of time an object is specified as non-dummy (i.e., it corresponds to a class modeled in class diagrams) to the total number of time the object appears in sequence diagrams. However, this approach showed similar (but somewhat inferior) result in the correlation analysis, thus we decided to report the result of LoD measurement at diagram level.

The definitions of the LoD measures from class- and sequence diagrams (i.e., CD_{aop} , CD_{asc} , SD_{obj} , and SD_{msg}) are aggregate measures. We prefer this aggregate measures to the individual LoD measures because the aggregate measures captures LoD at a level of granularity that allows us to still obtain meaningful variance in the data sets—this is particularly true when many of the individual metrics have a low variance. Further, because the aggregate measures are related to certain modeling aspects (e.g., objects, messages), we can meaningfully interpret the values of the aggregate measures.

Further, note that the direct metrics (e.g., $CD_{attrsig}$, SD_{par}) have the minimum and maximum values of 0 and 1 respectively. Consequently, the minimum value of the aggregate measures is 0 and the maximum value is the total number of the direct metrics. For example, the minimum and maximum values of SD_{msg} are 0 and 3 respectively.

6.4 Design of Study

In this section, we discuss the design of the study, which include the research questions, measured variables, and the analysis method.

6.4.1 Research Questions

Having discussed the LoD measures in class diagram and sequence diagram, the main research question with regard to the correlation between LoD and defect density can be further elaborated into more specific research questions:

1. Is there a significant correlation between LoD of classes in a UML model measured using class diagram LoD measures, i.e., CD_{aop} and CD_{asc} , and the defect density of the associated implementation?
2. Is there a significant correlation between LoD of classes in a UML model measured using sequence diagram LoD measures, i.e., SD_{obj} and SD_{msg} , and the defect density of the associated implementation?

In the subsection that follows, we discuss the dependent variable and confounding factors in this study.

6.4.2 Measured Variables

In this study, the independent variable is Level of Detail (LoD). In Section 6.3.2 we have defined several metrics to measure LoD.

We use defect density as the dependent variable to measure software quality. Before discussing the measurement of defect density, we need to underline three important concepts, namely: *findings*, *defects*, *change-sets*, and *faulty classes*. Findings refer to general problems reported in the bug tracking system. However, not all findings will be considered as defects because findings also contain entries not related to defects, such as requests for additional functionality. We define defects as errors or problems caused by incorrect implementations—that is, implementations that deviate from the specifications. A change-set is a set of source files that is modified in relation to fixing a defect. We refer to Java classes in change-sets—thus, Java classes that were modified to solve defects, as faulty classes.

Defect density measures the quality of software based on the number of defects found in a piece of software relative to its size. In our approach, defect density of an implementation class is measured from the number of times that particular class is corrected to solve distinct defects, divided by the size of the class (in kilo SLoC). It is important to point out that we could have been using defect-count (the number of defects per class) as the dependent variable. However, the variability of defect-count in our data set is small: 76 percent of the faulty classes contain only one defect (see Table 6.1). Using a dependent variable with low variability would have affected our ability to identify significant relationships between the LoD measures and the dependent variable.

6.4.3 Other Factors to Be Controlled

In addition to the main variables, in this study we identified and assessed two variables that might confound the results of the analysis, namely coupling between objects (CBO) [35] and McCabe's cyclomatic complexity metrics [84]. These variables are potential covariates because of their strong correlation with class fault-proneness (see for example in [121] and [71]). By incorporating these two variables we actually account for their effects on the variability in defect density.

We need to underline that the motivation of including the confounding factors are not to prove their correlations with defect density. Instead, it is to unveil the pure correlation between LoD and defect density by separating the effects of the confounding factors from that of LoD measures.

6.4.4 Analysis Method

To assess the correlation between the LoD measures and defect density, we used multiple linear regression. We performed multiple linear regression so that we can obtain standardized parameter estimates of each LoD measure, which indicate the total variance in defect density uniquely accounted for by each of the LoD measure. This way, the pure effect of each LoD measure on the variability in defect density can be observed.

In the multiple regression analyses, we use the stepwise selection (i.e., the backward elimination method) to select independent variables that have significant influences on the dependent variable. The backward elimination method starts with all independent variables included, and iteratively removes variables that have least impact on the predictive capability of the regression model. Backward elimination is preferred to the forward selection, which starts with no independent variable and iteratively incorporate variables that give significant improvement to the model, because the forward selection method is more likely to exclude a variable that has a significant effect but only when another variable is held constant (a.k.a suppressor effects) [49].

Note that in the statistical analyses, we use a significance level of 0.05 to indicate a true significance—that is, p-values below or equal to 0.05 ($p \leq 0.05$) are considered significant.

6.5 Case Study

The case study used in this chapter is the same as the one discussed in Chapter 4. Please refer to Chapter 4 for the description of the case study. In this section, we discuss data collection and processing that is applicable for this chapter.

6.5.1 Data Collection and Processing

Similar to the approach discussed in Chapter 4, to collect data about UML classes (hereafter referred to as *design classes*) and other metrics, the UML model first need to be exported

from the UML CASE tools into an XMI format. We then use the SDMetrics tool [4], to extract UML model information, such as design classes and other diagram elements, from the XMI file. We also use SDMetrics to define the LoD measures and calculate them from the XMI file. Furthermore, we use the CCCC (C and C++ Code Counter) tool to collect data about implementation classes (i.e., Java files) from source code and the code metrics (i.e., size, complexity, and coupling) of these classes.

Processing finding data mainly involves two steps. The first step is to obtain registered findings from the ClearQuest repository and store them in the analysis database. Finding registration contains textual information that explains the nature of each finding, which will be used to determine whether a finding can be regarded as a defect and of which type. The second step is to obtain change-sets from the versioning system (i.e., Rational Clearcase). To this aim, we use Clearcase's Perl interface (cc_perl), through which we can execute a script that automatically recovers change-set associated with every finding. Because the change-sets obtained are in a textual format and they also contain other information, text parsing is performed to mine data of the modified files (note that only Java files are taken into account, and each Java file represents a Java class). As mentioned earlier, this data of Java files that were modified to solve defects is then deemed as faulty classes.

Further, we determine the number of defects (defect-count) of each Java class based on the frequency by which a particular Java file is modified as part of the solving of a *distinct* finding. Hence, a Java class that is modified several times to solve the same finding is regarded as having a defect-count of only one.

Once processed, the above data is stored into a relational database. This database can be accessed via a web interface to enable remote collaboration for data collection and analysis.

Once the data of findings, design classes, implementation classes, and faulty classes are stored in the analysis database, the next step is to perform a matching between design classes and the implementation classes. This matching process is performed semi-automatically based on name and directory structure similarities. Likewise, the same matching process is also performed to match faulty classes and implementation classes. These matching processes allow us to establish links between design classes, implementation classes, and faulty classes. As a result, we can identify which of the faulty classes are modeled in which UML diagrams and, additionally, through their relationships with the implementation classes, their code metrics values could be determined.

It is worth mentioning that performing such matching processes are not trivial because often classes in the UML model are named differently in the implementation. Additionally, experience shows that for between 10% - 50% of the implementation classes, there exists a corresponding classes in the UML model (please refer to [94, 127]) for further discussion on model - code correspondence).

Besides the matching between design classes and the implementation classes, a manual matching is done between class instances in sequence diagrams and their corresponding implementation classes. As a result, for every modeled implementation class, the class- or sequence diagrams in which it is modeled could be identified. Thus, for classes from the implementation we can determine the LoD of the corresponding classes in the UML model based on how they are modeled in either class- or sequence diagrams. Figure 6.2 provides

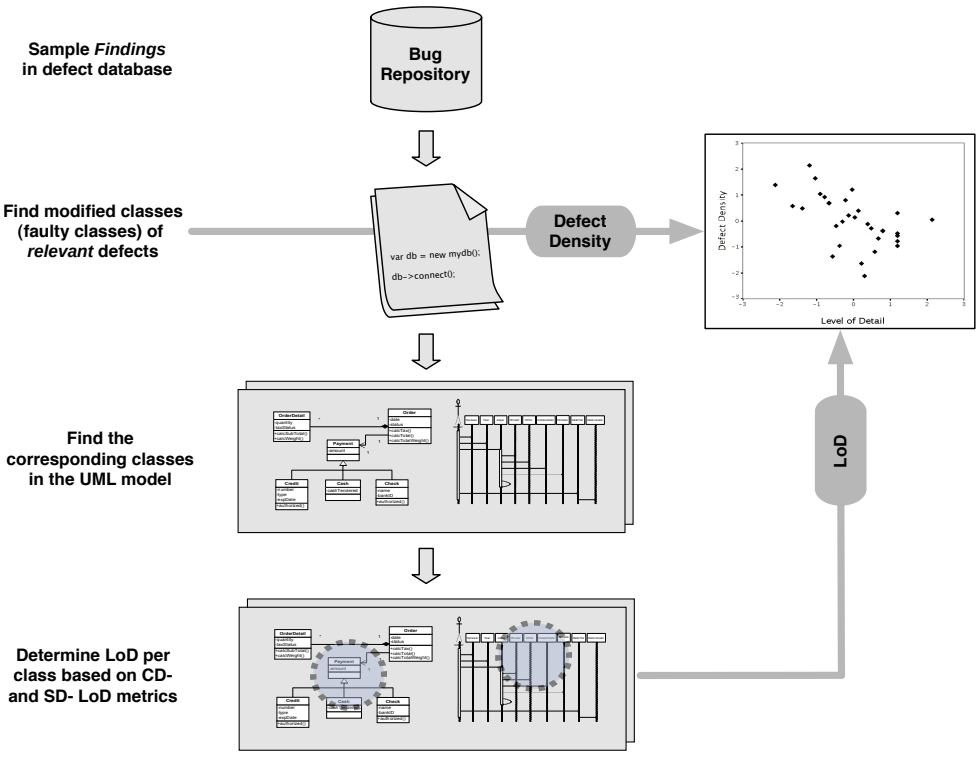


Figure 6.2: Overview of steps in collecting and processing data

an overview of the steps in data collection and data processing.

6.6 Data Analysis and Results

In this section, we discuss data analyses and findings. We start by explaining the analysis procedure.

6.6.1 Analysis Procedure

Before the main statistical analyses, manual analysis of findings had to be performed. This analysis is done to determine whether a finding could be regarded as defect and of which defect type. This analysis is very important because we will exclude findings that do not meet our definition of defect. In the following passages, we discuss both the finding analysis and statistical analysis performed in this study.

We regarded a finding as defect if it was registered due to explicit errors in the system or due to deviations from explicitly stated requirements. Hence, findings registered to incor-

porate additional functionality into the system were regarded as defect and were excluded from the analysis.

Besides excluding findings that are not defect, it is important to determine the defect type of every finding that has been regarded as defect. To this end, we categorize findings into several defect types based on the same defect taxonomy previously introduced in Chapter 4. For convenience, we recapitulate the defect taxonomy:

- **User Interface.** Defects related to static user interface layouts or caused by wrong or missing user interface navigation.
- **User Data Input/Output.** Defects related to missing or wrong data input/output from/to user interface.
- **Data Handling.** Defects caused by missing or wrong data handling, such as input data validation and session issues. Data access problems also belong to this category.
- **Computational.** Defects caused by missing or incorrect computation.
- **Logic/Algorithm.** Defects caused by missing or poor implementation of business rules or wrong formulation of conditions.
- **Process Flow.** Defects caused by missing or wrong process flows (e.g., incorrect order of operation execution).
- **Race Condition.** Defects caused by incorrect timing of events (e.g., unanticipated locking or synchronization).
- **Undetermined.** Defects do not belong to the above categories.

Once all the findings have been analyzed and categorized, findings that did not meet our definition of defects were excluded. Additionally, user interface defects and undetermined defects were excluded from the analysis because they rarely could have been avoided by introducing UML modeling. Having excluded those irrelevant findings and defect types, we are assured that we did not overstate defect-count of faulty classes due to the use of irrelevant findings/defects. Once we obtained the purified data, we proceeded with the statistical analyses.

6.6.2 Descriptive Statistics

The empirical data used in this study, including the defect data, is based on the latest version of the IPS' project data. Besides actuality reason, the latest version is chosen because most of the development activities took place in this version.

There are 1546 findings that we considered in this study. These findings are those reported during testing (i.e., unit test, system test, regression test, and integration test) and represent 60 percent of the total number of findings. The rest of the findings include those reported during review (771) and acceptance test (212). Out of the 1546 findings, only 566 are traceable to the modified source files. The fact that there are a large number of findings that is not traceable to the modified source files might be due to the following reasons. First, there are findings that were solved without modifying source files, which

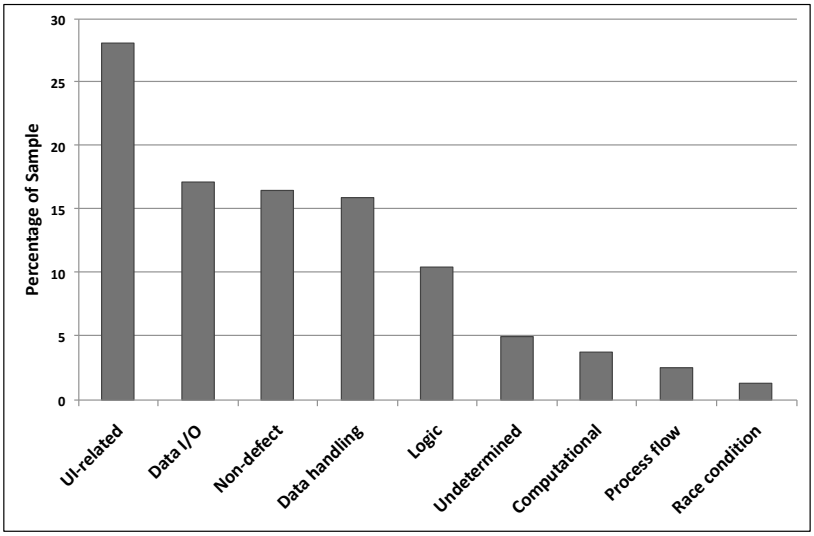


Figure 6.3: Defect type distribution of the sampled findings

include changes in the database or application server. Second, it is possible that a finding was solved indirectly, i.e., by solving other findings. Finally, it is often the case that findings were rejected for some reasons, for instance because they could not be reproduced. In these cases, no source file was corrected to solve the findings.

Since defect-source traceability is a prerequisite for the analyses, findings without this traceability had to be excluded; thus, the population for the analysis is 566 findings. Out of these 566 findings a random sampling is performed. The sampling is performed by assigning a random number to each finding and then sorting the findings based on the random numbers. The sample is taken from the first 164 findings from the sorted list. The size of the sample is mainly constrained by the availability of resources to perform defect analysis.

Defect Type Distribution

As discussed previously, we performed manual inspection to determine whether findings registered in the defect repository can be considered as defects. This inspection also allowed us to determine the type of defect of each finding. Figure 6.3 shows all of the 164 findings that were analyzed—prior to the exclusion of irrelevant findings, and categorized based on their defect types.

As can be seen from Figure 6.3, a large number of the findings falls into the category of UI-related defects (28 percent), followed by user data I/O (17 percent) and data handling (16 percent)—the rest being equal to or lower than 10 percent of the sample size. Also note that a considerable number of findings falls into non-defect category (27 percent). In this respect, we have observed that many of the findings categorized into non-defect are those

related to change requests. Additionally, five percent of the analyzed findings could not be categorized into any of the defect categories(undetermined).

Except for the UI-related, undetermined, and non-defect, all defect types are considered for further analyses. UI-related defects are excluded because they are of a type for which UML modeling could not have prevented their occurrence—this is particularly true in the IPS project because UI-related features were not modeled. The fact that many of the defects fall in the defect category that is not modeled, i.e., UI-related, may already signify the substantial effect of UML on the quality of the implementation. After having excluded UI-related, undetermined, and non-defect categories, we are left with 83 usable defects for further analyses.

Faulty Classes

There are 122 faulty classes that were modified to solve the 83 defects. This number is slightly smaller than the number reported in [98] (i.e., 134) because in the previous study we overlooked Java classes that are actually test cases, but were not explicitly named as such. However, these classes are all not modeled in the UML model, hence excluding them would not affect the results of the analysis.

Of 122 faulty classes, only 37 are modeled as design classes. However, we found that five faulty classes are modeled as design classes, but they are not used neither in class- nor sequence diagrams—it seems that these classes were created, but were later removed from the diagrams in which they were initially displayed. We considered these faulty classes as not modeled because it is unclear whether their corresponding design classes were ever consulted during the implementation. Therefore, in the end we had 32 faulty classes that are modeled either in class- or sequence diagrams

The matching process between faulty classes and design classes is performed semi-automatically based on name and directory-structure similarity. The profile of the 122 faulty classes with respect to their presence in the UML model is as follows.

- 23 classes of the implementation are modeled as classes in the class diagrams.
- 30 classes of the implementation are modeled as objects in sequence diagrams.
- 21 classes of the implementation are modeled as both classes in class diagrams and objects in sequence diagrams.
- 85 classes of the implementation are not modeled at all.

Table 6.1 shows the distribution of defects across the faulty classes. Note that data in Table 6.1 also includes implementation classes that are not modeled. As Table 6.1 shows, more than 75 percent of the faulty classes (93 of them) contain one defect, and slightly more than 13 percent (17 of them) contain two defects. The rest of the classes (12 of them) contain between three to seven defects. Figure 6.4 visualizes the same data using a histogram.

Box-plots in Figure 6.5 provide an overview of the 122 faulty classes in terms of defect density, complexity, and coupling. The box-plots show the presence of outliers and extreme

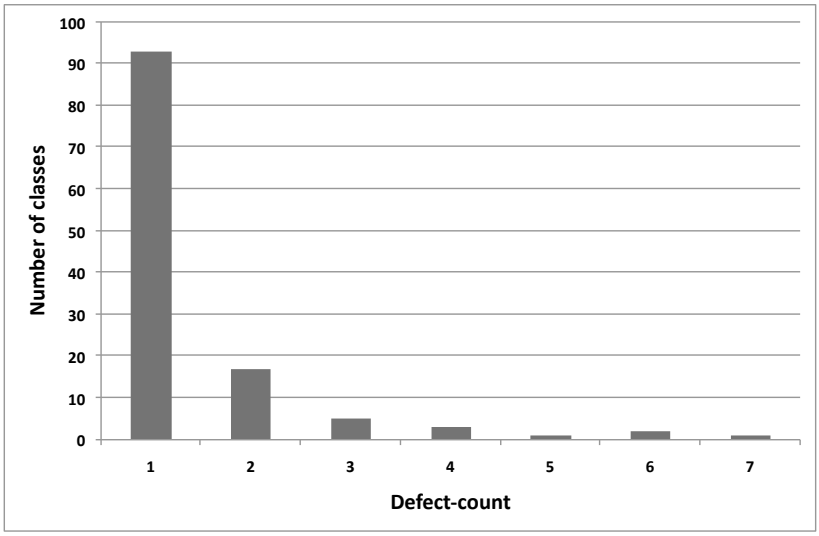


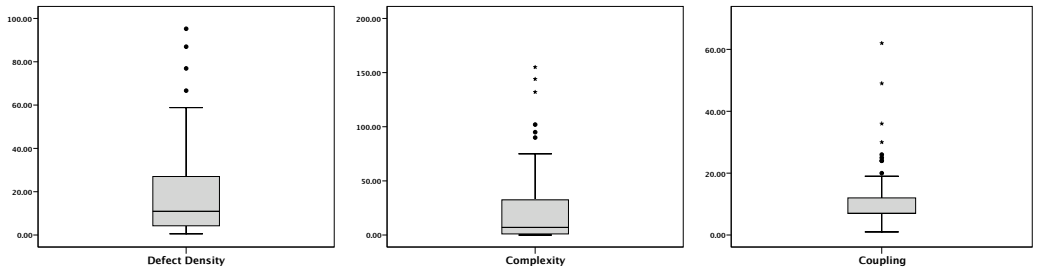
Figure 6.4: Histogram of defect distribution across the 122 faulty classes

Table 6.1: Distribution of defects across faulty classes

# of classes	Percentage	Defect-count
93	76.20	1
17	13.90	2
5	4.10	3
3	2.50	4
1	0.80	5
2	1.60	6
1	0.80	7
122	100.00	178

values in the data, particularly for the complexity and coupling metrics. However, after analyzing each outlier and extreme value, we found that they are valid data points, thus we did not have a strong reason to remove them from the data sets. Note that we excluded several highly extreme values from the complexity and coupling box-plots because including them would compress/narrow the box-plots.

Table 6.2 and 6.3 provide descriptive statistics of faulty classes that are modeled using UML—that is, 23 and 30 faulty classes in class- and sequence- diagrams respectively. Note that the maximum values for complexity and coupling in both tables are not shown in Figure 6.5. These values are the extreme values that are excluded from the box-plots to improve readability. From the information in both tables we can learn some characteristics of the measured variables. For example, in Table 6.3 we can see that SD_{obj} has a relatively low variance, which is indicated by its low standard deviation. Further, the mean value of SD_{obj} is very close to its maximum value (also note the median value, which is equal to the maximum). These numbers indicate that most class objects that are modeled in sequence diagrams have high object detailedness—that is, mostly they are named and corresponded



(a) Defect Density (b) McCabe's Complexity (c) Coupling

Figure 6.5: Box-plots of defect density, complexity, and coupling of the 122 faulty classes

Table 6.2: Descriptive statistics of implementation classes modeled in class diagrams

Measures	N	Med	Mean	SDev	Min	Max
CD_{aop}	23	0.50	0.47	0.48	0.00	1.00
CD_{asc}	23	0.00	0.44	0.49	0.00	1.00
Coupling	23	16.00	22.43	24.14	3.00	119.00
Complexity	23	66.00	63.86	70.09	0.00	297.00
Size (KSLoC)	23	0.54	0.77	1.08	0.05	5.26
Defect Density	23	3.00	6.05	6.51	0.56	21.74

Table 6.3: Descriptive statistics of implementation classes modeled in sequence diagrams

Measures	N	Med	Mean	SDev	Min	Max
SD_{obj}	30	2.00	1.97	0.03	1.90	2.00
SD_{msg}	30	1.74	1.70	0.54	0.61	2.56
Coupling	30	15.00	22.16	22.84	7.00	119.00
Complexity	30	54.50	67.40	85.04	0.00	366.00
Size (KSLoC)	30	0.40	0.87	1.51	0.02	7.01
Defect Density	30	3.15	10.48	17.84	0.56	86.96

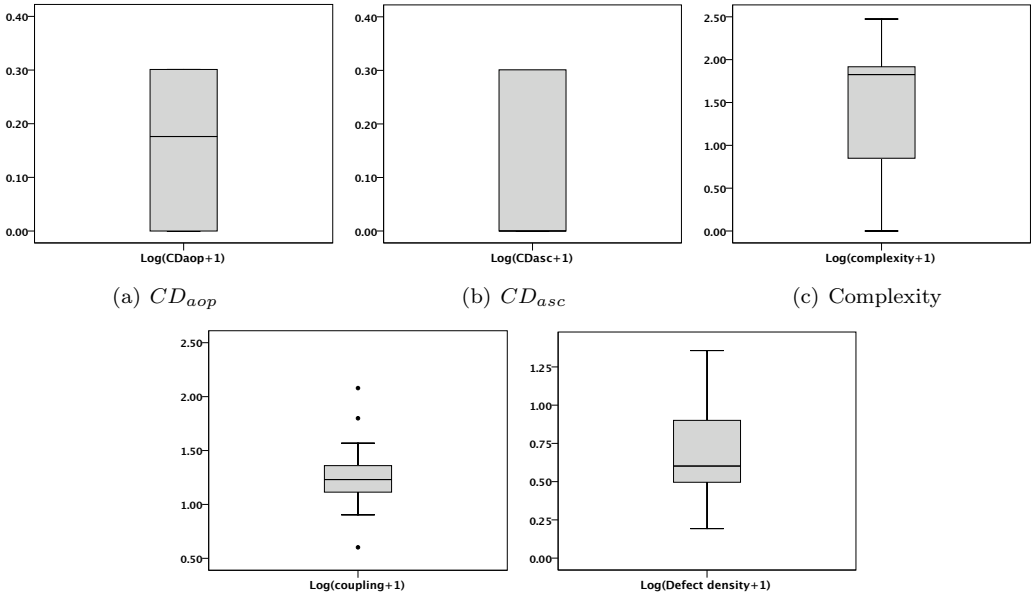
to real design classes in class diagrams.

6.6.3 Correlation Analyses between LoD and Defect Density

In this section regression analyses will be performed to investigate the relation between the LoD measures and defect density. We start by assessing the independent variables used in the analyses.

Data Transformation

We performed *log* transformation on all variables used in the analysis. Although linear regression does not assume a normal data distribution (normality assumption), data trans-



(a) CD_{aop} (b) CD_{asc} (c) Complexity

(d) Coupling (e) Defect Density

Figure 6.6: Box-plots of the 23 faulty classes modeled in class diagrams (after log transformation)

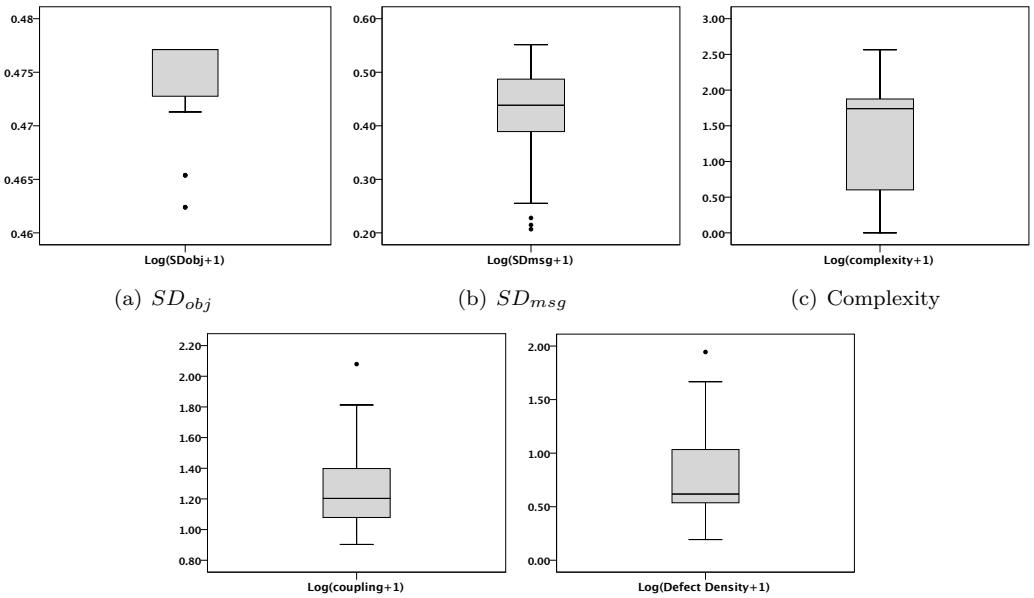
formation generally help reduce the impact of outliers [49]. For a variable k, the transformed variable $k' = log(k+1)$. Note that we added 1 to the variable because the data sets contained zero values.

Log transformation is useful for reducing the impact of large values. As such, it brings large values closer to the centre of the distribution. Having performed this transformation, we expect to reduce a positively skewed data, as shown in Figure 6.5. The box-plots of the log-transformed variables of the faulty classes modeled in class diagrams and sequence diagrams are provided in Figure 6.6 and 6.7 respectively.

Therefore, please keep in mind that variable values discussed in the rest of this chapter are the results of log transformations.

Multicollinearity Diagnostic

Prior to the analyses, we needed to assure that there is no significant correlation amongst the independent variables. When such strong correlations exist between two or more variables then we face a multicollinearity issue. The presence of multicollinearity might threat the validity of multiple regression analysis: 1) it limits the size of R; 2) it makes assessing the importance of the individual predictors difficult as they are highly correlated; and 3) it might result in unstable predictor equations [49].



(a) SD_{obj} (b) SD_{msg} (c) Complexity

(d) Coupling (e) Defect Density

Figure 6.7: Box-plots of the 30 faulty classes modeled in sequence diagrams (after log transformation)

Table 6.4: Correlation between independent variables of class diagram LoD (Spearman's)

	CD_{aop}	CD_{asc}	MCC	CBO
CD_{aop}	1.000	.388	.088	.406
Sig.	.	.067	.689	.055
CD_{asc}		1.000	.418*	.463*
Sig.		.	.047	.026
MCC			1.000	.160
Sig.			.	.467
CBO				1.000
Sig.				.

* indicates $p \leq 0.05$ (2-tailed)

To detect multicollinearity, we performed correlation analyses amongst the independent variables. Table 6.4 and 6.5 provide the results of the correlation analyses for class- and sequence- diagram LoD measures respectively. Please note that the correlation analyses are done on the log-transformed data sets.

Table 6.5: Correlation between independent variables of sequence diagram LoD (Spearman's)

	SD_{obj}	SD_{msg}	MCC	CBO
SD_{obj}	1.000	-.224	-.197	.120
Sig.	.	.234	.297	.527
SD_{msg}		1.000	.350	.240
Sig.		.	.058	.202
MCC			1.000	.300
Sig.			.	.108
CBO				1.000
Sig.				.

Table 6.6: Results of univariate regression for class diagram LoD measures

Model	Unstdized. Coef.		Std. Coef.	t	Sig.	Model Summary		
	B	Std. Error	Beta			R	R^2	Adjusted R^2
CD_{aop}	.315	.508	.134	.621	.542	.134	.018	-.029
CD_{asc}	-.617	.489	-.266	-1.263	.220	.266	.071	.026
Complexity	-.250	.076	-.583	-3.289	**.003**	.583	.340	.309
Coupling	-.518	.229	-.443	-2.267	**.034**	.443	.197	.158

As we can see in the tables, there is no substantially high correlation amongst the independent variables. Significant correlations exist between CD_{asc} and complexity, and between CD_{asc} and coupling. However, the magnitude of these correlations is not substantially large, and thus multicollinearity is not considered a threat. Consequently, in the multiple regression analyses we include all the independent variables.

Using Class Diagram LoD Measures

This analysis aims to explore if there is a significant correlation between class diagram LoD measures, namely CD_{aop} and CD_{asc} , and defect density. To this aim, first we explore the contributions of all independent variables to the dependent variable by means of simple regression analysis. The result is shown in Table 6.6.

The most important point to note in Table 6.6 is that none of the class diagram LoD measures has significant effect on defect density. On the other hand, both complexity and coupling have significant effects on defect density (the p-values are 0.003 and 0.034 respectively). Additionally, we can see in the table that complexity has the strongest influence on defect density with $R^2 = 0.34$—this means that 34 percent of variability of defect density is explained by class complexity. Additionally, the b-values indicate the nature of the relationships between defect density and the predictors. As shown in the table, both complexity and coupling have negative correlations with defect density.

To assess the unique variance of defect density that is explained by the class diagram LoD measures, we performed a multiple regression analysis in which we used CD_{aop} , CD_{asc} , complexity, and coupling as predictors (note that we use the backward elimination method to select the best predictors). By including class complexity and coupling in the regression analysis, we actually account for their effects on defect density.

Having performed multiple regression analysis using the backward elimination method, we obtain three significant predictors of defect density, namely CD_{aop} , complexity, and coupling—CD_{asc} is disqualified as a significant predictor (see Table 6.7). It is interesting to note that CD_{aop} , which had no significant contribution to defect density in the univariate analysis, is selected as one of the significant predictors in the multivariate analysis. Further, we can see that the b-value (coefficient) of CD_{aop} is positive, which suggests that as the value of CD_{aop} increases, the value of defect density also increases. In the contrary, both complexity and coupling have negative correlations with defect density.

Although the b-values are useful for assessing the contribution of the individual predictors, the standardized coefficients (beta) is easier to interpret because they are not dependent on the units of measurement of the predictors—that is, they are all measured in standard deviation units. As such, the standardized coefficients are directly comparable across predictors, thus providing better insights about the importance of a predictor in a regression model. Having said that, we can see in Table 6.7 that while CD_{aop} , complexity, and coupling are all significant predictors, the standardized coefficients (beta) suggest that class complexity has the highest contribution in the regression model.

The column labeled Correlations in Table 6.7 provides correlation coefficients between the predictors and the dependent variable. In this analysis, we are mainly interested in part (semi-partial) correlation coefficient, which indicates the unique variance in the dependent variable that is explained by a particular predictor—that is, the variance explained or shared by other predictors is removed. As such, the squared value of a part correlation represents the percentage of total variance in the dependent variable uniquely accounted for by a particular predictor, and not by the other predictor(s). Hence, we can calculate from Table 6.7 that CD_{aop} has nearly 17 percent unique contribution on the variability of defect density. On the other hand, complexity and coupling contribute 34 and 22 percent respectively to the variability of defect density. Note that standardized regression coefficients (beta) are essentially part (semi-partial) correlations, hence their values are quite similar.

Another point to note from the results is that the regression model significantly predicts defect density ($R^2 = 0.63; p \leq 0.001$). The value of the r-square indicates the predictive capability of the model—that is, it explains 63 percent the variability of defect density. However, as the main goal of the regression analyses is to assess the contribution of the LoD measures, we are not concerned so much about the predictive capability of the overall regression model. What is more important for us is to assess whether CD_{aop} and CD_{asc} significantly correlate with defect density after the effects of class complexity and coupling have been accounted for.

Additionally, we need to consider multicollinearity amongst the predictors because highly correlated predictors cause the regression coefficient estimates to be unreliable. To assess multicollinearity, we can use the VIF (variance inflation factor) values. It is suggested that

Table 6.7: Results of multivariate regression for class diagram LoD measures

Model	Unstdized. Coef.		Std. Coef.	t	Sig.	Correlations			Collinearity	
	B	Std. Error	Beta			Zero-order	Partial	Part	Tolerance	VIF
(Constant)	1.645	.218		7.548	.000					
CD_{aop}	1.050	.355	.447	2.958	**.008**	.134	.562	.412	.846	1.182
Complexity	-.258	.062	-.600	-4.184	**.001**	-.583	-.692	-.582	.940	1.064
Coupling	-.585	.175	-.501	-3.351	**.003**	-.443	-.609	-.466	.867	1.154

Model Summary: $R^2 = 0.63; p \leq 0.001$.

Table 6.8: Results of multivariate regression analysis for sequence diagram LoD measures

Model	Unstdized. Coef.		Std. Coef.	t	Sig.	Correlations			Collinearity	
	B	Std. Error	Beta			Zero-order	Partial	Part	Tolerance	VIF
(Constant)	2.425	.275		8.823	.000					
SD_{msg}	-1.816	.535	-.394	-3.395	**.002**	-.622	-.554	**-.365**	.858	1.166
Complexity	-.252	.063	-.469	-3.970	**.001**	-.686	-.614	-.426	.827	1.209
Coupling	-.419	.185	-.260	-2.260	**.032**	-.513	-.405	-.243	.871	1.148

Model Summary: $R^2 = 0.70; p \leq .001$.

an average VIF substantially greater than 1 indicates the presence of multicollinearity [20]. However, as the average VIF in Table 6.7 is only 1.133, we are assured that multicollinearity is not a problem.

Having discussed the above results, we can conclude the following points. First, neither CD_{aop} nor CD_{asc} has a significant effect on defect density when used as a sole predictor. Second, CD_{aop} turns out to be a significant predictor when it is combined with complexity and coupling metrics. Additionally, we observe that CD_{aop} has a positive correlation with defect density—that is, the higher CD_{aop} , the higher the defect density. This result contradicts the presumed negative correlation between LoD and defect density. A plausible explanation for this result is that classes with a large number of their attribute signatures and operation parameters being defined are inherently more complex. These classes are likely to deal with complex computations or complex data manipulations—complexity aspects that are not measurable using McCabe's complexity metric. As a result, classes with higher CD_{aop} tend to be more defect prone. Finally, we see that both coupling and complexity metrics are significant predictors of defect density. Moreover, we learn that both coupling and complexity metrics have negative correlations with defect density.

As for complexity and coupling, their negative correlations with defect density can be partly explained by the fact that both metrics are strongly correlated with class size. Naturally, class size will have a negative correlation with defect density as it serves as the denominator of defect density (there is a negative correlation between X/Y and Y)—please refer to [112] for further discussion on this matter. While we agree that the significant correlation between either complexity or coupling and defect density is partly due to a mathematical artifact (i.e., these three variables share a common factor: size), other factors might also contribute to the correlations. For example, classes with high coupling and complexity could have been treated more carefully during the construction phase. As a result, classes with high coupling or complexity might have less defect density. It is precisely due this reason that we account for complexity and coupling as covariates in the analysis.

Using Sequence Diagram LoD Measures

In this section, we analyze the correlation between sequence diagram LoD measures and defect density. SD_{obj} and SD_{msg} are used to measure the LoD of the 30 faulty classes that appear as objects in sequence diagrams. The descriptive statistics of the 30 faulty classes is presented in Table 6.3.

As with the previous analysis, for this analysis we first assess the effects of sequence diagram LoD measures on defect density by means of simple regression. As shown in Table 6.9, all of the predictors except SD_{obj} have significant contributions to defect density. In particular, we can see that SD_{msg} is highly significant ($p \leq 0.001$) and it contributes 38 percent ($R^2 = 0.386$) to the variability of defect density. Likewise, complexity is also a highly significant predictor and it has the highest contribution to the variability of defect density (47 percent). Although somewhat lower than the other predictors, coupling contributes 26 percent of the variability in defect density.

We subsequently perform a multiple regression analysis to assess the pure contributions

Table 6.9: Results of univariate regression for sequence diagram LoD measures

Model	Unstdized. Coef.		Std. Coef.	t	Sig.	Model Summary		
	B	Std. Error	Beta			R	R^2	Adjusted R^2
SD_{obj}	24.900	15.029	.299	1.657	.109	.299	.089	.057
SD_{msg}	-2.867	.683	-.622	-4.200	**.000**	.622	.386	.365
Complexity	-.369	.074	-.686	-4.990	**.000**	.686	.471	.452
Coupling	-.826	.261	-.513	-3.164	**.004**	.513	.263	.237

of the individual predictors—though we are particularly interested in the pure effect of SD_{msg} on defect density. As with the previous multiple regression analysis, we perform the backward elimination method to select influential predictors. Running the backward elimination method with all independent variables included (i.e., SD_{obj} , SD_{msg} , complexity, and coupling) results in a regression model that excludes SD_{obj} as predictor. The result of the multiple regression analysis is provided in Table 6.8.

The most important point to mention in Table 6.8 is the fact that SD_{msg} remains a significant predictor for defect density ($p \leq 0.01$), after controlling for the effects of complexity and coupling. Furthermore, by calculating the r-square of the part correlation (-0.365^2), we learn that 13 percent of the total variance in defect density is uniquely accounted for by SD_{msg} . Note that this amount of variance explained by SD_{msg} is one-third that of observed in the simple regression. As such, this result actually tells us that two-third of the variance in defect density that is accounted for by SD_{msg} (in the simple regression analysis) is also shared by complexity and/or coupling. On the other hand, complexity and coupling account for 18 and 5 percent respectively of the variability of defect density. As with the standardized coefficients (beta), the part correlation indicates the importance of a particular predictor in a prediction model. In this regard, again, complexity is the most influential predictor in the regression model. Additionally, we can see that the b-values (coefficient) of SD_{msg} , complexity, and coupling are all negative—this means, as the values of these metrics increase, defect density decreases.

Table 6.8 also shows that the regression model has a fairly good predictive capability ($r^2 = 0.70$)—that is, the model explains 70 percent of the variability of defect density. Compared to the regression model created using the class diagram LoD measures, this model has seven percent higher predictive capability. A final remark concerns the multicollinearity issue. We can see in the table that the average VIF of the predictors is only slightly above 1. Therefore, for this multiple regression analysis we also do not consider multicollinearity as a problem that might hinder accurate coefficient estimates.

The primary conclusion that we can draw from the above analyses is that SD_{msg} is a significant predictor of defect density. In particular, we have learnt that the effect of SD_{msg} on defect density after controlling for the effects of complexity and coupling is roughly one-third that of when neither complexity nor coupling is accounted for. As discussed earlier, this phenomenon indicates that to some degree there is an interaction between the variances of SD_{msg} , complexity, and coupling in the dependent variable. Therefore, by controlling for the effects of complexity and coupling we essentially have discounted the shared variance, and thus unveiling a more genuine effect of SD_{msg} on defect density.

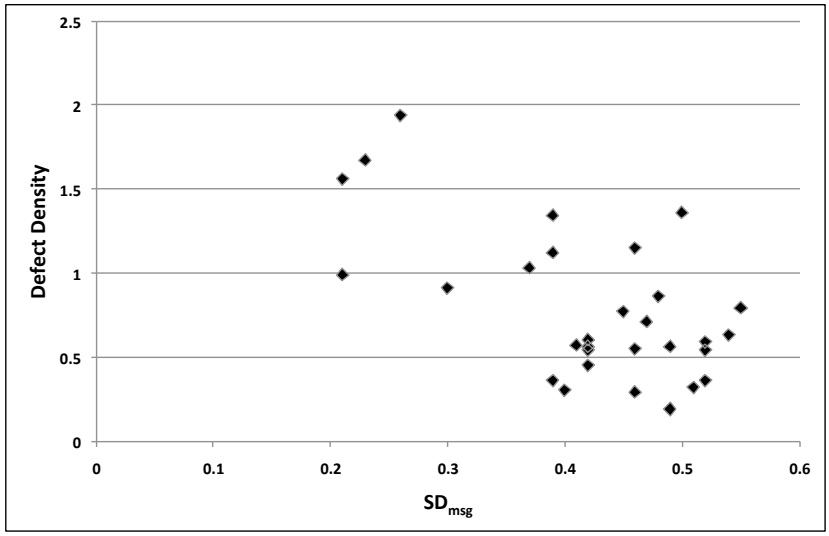


Figure 6.8: Scatterplot showing the correlation between SD_{msg} and defect density

Table 6.10: Results of univariate regression for class diagram LoD measures — NS-OFI data set

Model	Unstdized. Coef. B	Unstdized. Coef. Std. Error	Std. Coef. Beta	t	Sig.	Model Summary R	Model Summary R^2	Model Summary Adjusted R^2
CD_{aop}	.321	.365	.129	.880	.383	.129	.017	-.005
CD_{asc}	-.564	.343	-.236	-1.647	.106	.236	.056	.035
Complexity	-.228	.057	-.510	-4.016	**.000**	.510	.260	.244
Coupling	-.359	.181	-.281	-1.988	.053	.281	.079	.059

Additional Analyses Using Larger Data Sets

We perform additional analyses using data sets in which no defect sampling is performed as to verify whether we would still obtain similar results. Additionally, we considered including non-faulty classes in the analyses. However, including these classes has led to violation of linearity assumption because non-faulty classes have zero variance of defect density. Therefore, in this section we provide only the results of analysis in which we disregard defect types of the faulty classes. For brevity, we refer to the data sets as NS-OFI (No Sampling - Only Faulty Included). The descriptive statistics of the NS-OFI data sets for class- and sequence diagrams LoD measures are given in Appendix C.

Table 6.10 and 6.12 provide the results of the univariate and multivariate analyses of the class diagram LoD measures using NS-OFI data set. The most important point to note from both tables is that the correlation between CD_{aop} and defect density in both univariate and multivariate analyses using NS-OFI data set is consistent with the main analysis discussed previously. In particular, we have seen that there is no significant correlation between defect density and CD_{aop} in the univariate analysis. However, CD_{aop} turns out to be a significant

Table 6.11: Results of univariate regression for sequence diagram LoD measures — NS-OFI data set

Model	Unstdized. Coef. B	Unstdized. Coef. Std. Error	Std. Coef. Beta	t	Sig.	Model Summary R	Model Summary R^2	Model Summary Adjusted R^2
SD_{obj}	.287	6.744	.006	.043	.966	.006	.000	-.017
SD_{msg}	-1.777	.526	-.403	-3.381	**.001**	.403	.162	.148
Complexity	-.366	.057	-.639	-6.389	**.000**	.639	.409	.399
Coupling	-.402	.238	-.215	-1.688	.097	.215	.046	.030

predictor in the multivariate analysis after it is combined with McCabe's complexity metric. Also note that the nature of the correlation is also the same—that is, an increase in CD_{aop} corresponds to an increase in defect density.

Similarly, Table 6.11 and 6.13 show the results of univariate and multivariate analysis of the sequence diagram LoD metrics. The result of the univariate analysis shows that SD_{msg} and complexity are significantly correlated with defect density. Furthermore, it is interesting to note that the multivariate analysis also qualifies SD_{msg} and complexity as significant predictors of defect density. If we consider the results in Table 6.9 and 6.8, we can conclude that the correlation between SD_{msg} and defect density is consistent in the two data sets used in the analysis.

Having done the above analyses, we can conclude that the finding concerning the significance of CD_{aop} and SD_{msg} as predictors of defect density is not restricted to the first data set used in this study (i.e., when defect sampling is employed)—that is, the results is generalizable to a larger data set in which no defect sampling is performed. However, we should underline that the pure contributions (part correlation) of both CD_{aop} and SD_{msg} are larger in the multivariate analyses using the data set in which defect sampling is done. This result might indicate that noise in the data set is lower due to the sampling method used.

6.6.4 The Functional Form of LoD - Defect Density Relationship

Performing a linear regression analysis assumes that the relation between the independent and dependent variables is a linear one. When considering this linear relation between SD_{msg} and defect density, it is suggested that after increasing the LoD of the UML model up to a certain point, we can achieve zero defect density for implementation classes. However, in reality this will never be the case. A more reasonable relationship is actually a non-linear one: increasing LoD will never lead to zero defect density—instead, after a certain point, increasing LoD will not effectively reduce defect density (diminishing returns). In this regard, non-linear functions (e.g., polynomial, logistic, exponential) might be more appropriate in explaining the relationship between LoD and defect density.

To understand the form of the non-linear relationship between SD_{msg} and defect density, we perform curve estimation analyses. Curve estimation is useful for assessing relationships between two variables that are not necessarily linear. Scatter-plots in Figure 6.9 show curve fitting on the data set using Logarithmic, Inverse, Quadratic, and Cubic functions respectively. Furthermore, in Table 6.14, the results of statistical analyses (i.e., F test of

Table 6.12: Results of multivariate regression for class diagram LoD measures — NS-OFI data set

Model	Unstdized. Coef.		Std. Coef.	t	Sig.	Correlations			Collinearity	
	B	Std. Error	Beta			Zero-order	Partial	Part	Tolerance	VIF
(Constant)	1.408	.069		20.277	.000					
CD_{aop}	.996	.317	.399	3.147	**.003**	.129	.425	.365	.837	1.195
Complexity	-.300	.057	-.671	-5.285	**.000**	-.510	-.619	-.614	.837	1.195

Model Summary: $R^2 = 0.39; p \leq 0.001$.

Table 6.13: Results of multivariate regression analysis for sequence diagram LoD measures — NS-OFI data set

Model	Unstdized. Coef.		Std. Coef.	t	Sig.	Correlations			Collinearity	
	B	Std. Error	Beta			Zero-order	Partial	Part	Tolerance	VIF
(Constant)	2.053	.185		11.101	.000					
SD_{msg}	-1.144	.435	-.259	-2.632	**.011**	-.403	-.327	**-.251**	.938	1.066
Complexity	-.329	.056	-.575	-5.833	**.000**	-.639	-.608	-.557	.938	1.066

Model Summary: $R^2 = 0.47; p \leq .001$.

Table 6.14: Results of curve estimations between SD_{msg} and defect density

Function	Model Summary					Parameter Estimates			
	R-square	F	df1	df2	p	Constant	b1	b2	b3
Linear	.386	17.637	1	28	.000	1.997	-2.867		
Logarithmic	.415	19.885	1	28	.000	-.152	-1.050		
Inverse	.423	20.562	1	28	.000	-.098	.348		
Quadratic	.442	10.698	2	27	.000	3.406	-10.970	10.750	
Cubic	.446	10.871	2	27	.000	2.988	-7.166	.000	9.610

model fit and parameter estimates) showing how good each function fits the data set is given. We actually experimented using other functions such as exponential and logistic functions, but here we show only functions with r-square values greater than that of the linear function.

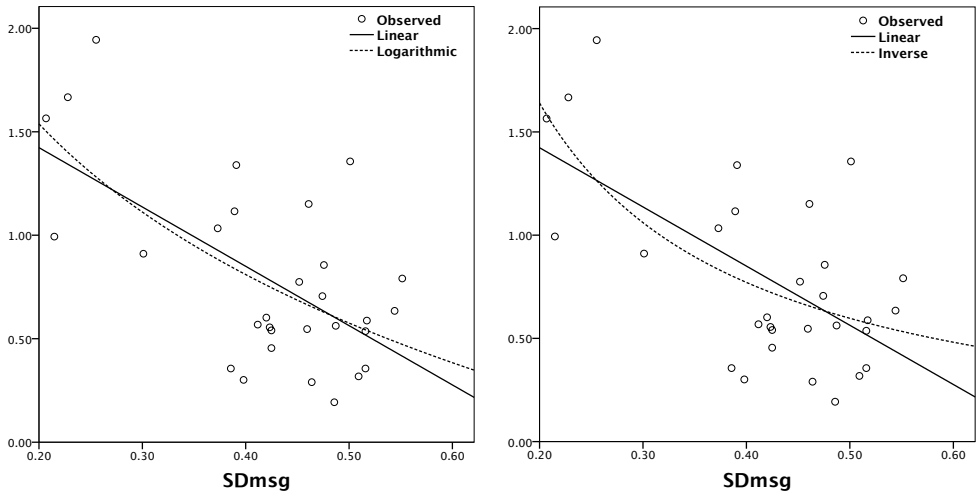
As shown in Table 6.14, the significance value $p \leq 0.05$ suggests that the variation explained by each function (model) is statistically significant (i.e., not due to chance). The R-square indicates the strength of the relationship between the observed and predicted values of defect density. We can see in the table that Cubic function predicts defect density better than the rest of the functions. However, in general there is no substantial difference of R-square amongst logarithmic, inverse, quadratic, and cubic functions.

From the functions in Table 6.14, we can see that the logarithmic function ($f(x) = \beta_0 + \beta_1.ln(x)$) generally has the same nature as the linear model: an increase in SD_{msg} corresponds to a decrease in defect density at a relatively similar rate. The quadratic ($f(x) = \beta_0 + \beta_1.x + \beta_2.x^2$) and cubic ($f(x) = \beta_0 + \beta_1.x + \beta_2.x^2 + \beta_3.x^3$) functions, on the other hand, suggest that after a certain point an increase in SD_{msg} will increase defect density. This is indicated by the positive values of b2 and b3 in the quadratic and cubic function respectively. The inverse function ($f(x) = \beta_0 + \beta_1/x$), however, shows a different type of relationship. After a certain point, an increase in SD_{msg} will not reduce defect density substantially—that is, defect density decreases at a slower rate.

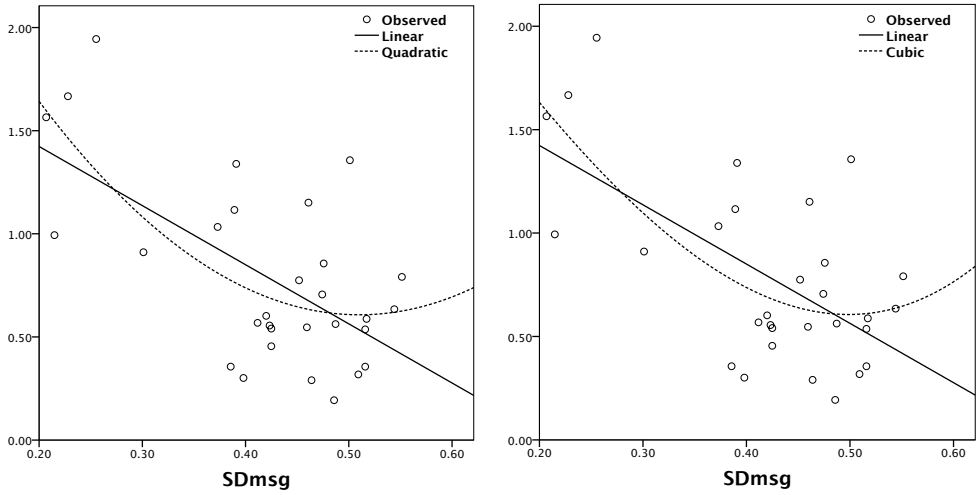
Considering the results, the three functions (i.e., inverse, quadratic, and cubic) suggest plausible explanations about the form of relationship between SD_{msg} and defect density. The inverse function, however, seems to support our assumption about the relation between LoD and defect density better. More research using additional project data is therefore needed in order to confirm whether we can see a typical form of relationship between the two variables.

6.6.5 The Relationship between LoD and Defect-fixing Effort

In Chapter 4, we have discussed the effect of using UML on the effort of fixing defects. The result shows that faulty behaviors that are modeled using sequence diagrams require significantly less effort to be fixed. With respect to LoD, it is therefore interesting to investigate whether there is a relationship between the LoD of sequence diagrams, which are related to faulty behaviors, and the effort spent on fixing defects. To this aim, we perform an additional analysis by reusing the data set of Chapter 4, but taking account



(a) Logarithmic (b) Inverse



(c) Quadratic (d) Cubic

Figure 6.9: Estimations of the form of functional relationship between SD_{msg} and defect density

only defects that are modeled using sequence diagrams.

Having taken into account only those defects that are modeled using sequence diagrams, we have a remaining of 34 data points to be used in the analysis. For each defect, we calculate the average object detailedness (ObjLoD) and message detailedness (MsgLoD) of

Table 6.15: Correlation between averaged MsgLoD and FixEffort (Spearman's)

	FixEffort
Averaged MsgLoD	.063
Significance (2-tailed)	.721

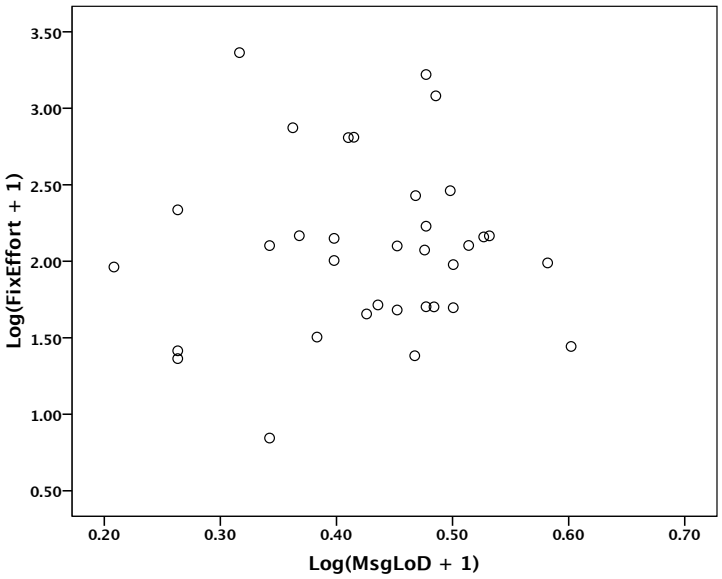


Figure 6.10: Scatterplot showing the correlation between MsgLoD and FixEffort

sequence diagrams in which the defective scenario/behavior are modeled. Subsequently, we perform a correlation analysis between both averaged ObjLoD, averaged MsgLoD, and FixEffort.

In Table 6.15 we provide the result of the correlation analysis. Notice that we only show the correlation analysis between averaged MsgLoD and FixEffort. We exclude ObjLoD because it has zero variance—performing a correlation analysis between two variables has no value if one of the variables has zero variance. The result in Table 6.15 is based on log-transformed variables (e.g., log-transformed FixEffort = log(FixEffort + 1)).

As shown in Table 6.15, we do not observe a significant correlation between message detailedness in sequence diagrams and defect-fixing effort (the significance value $p > 0.05$). A possible explanation for this result is that level of detail in sequence diagrams may not be as much influential for assisting maintenance-like activities as for guiding correct implementation of a system. Assuming that most developers remain the same during a software development project (as it was the case in the IPS project), the engineers who perform defect fixing would have been very well informed about details (both implicit and explicit)

in the sequence diagrams based on their exposure to them during the analysis, design, and implementation phases. As such, problems related to LoD in the sequence diagrams may have been clarified earlier, and thus they do not affect the defect fixing activities substantially. The scatterplot in Figure 6.10 visualizes the correlation between averaged MsgLoD and FixEffort (based on a log-transformed data set).

6.7 Discussion

In this section we discuss the interpretation of the results and their implications. Additionally, we discuss some limitations of this study.

6.7.1 Interpretation of Results

Correlation analyses between LoD measures and defect density have shown that SD_{msg} and CD_{aop} are significantly correlated with defect density in the implementation. However, the nature of the correlation between those two LoD measures and defect density is not the same: SD_{msg} has a negative correlation with defect density, CD_{aop} positive. Additionally, these results remain consistent when we use a larger data set (i.e., NS-OFI) in the analyses. The effects of class complexity and coupling also have been accounted for.

The fact that SD_{msg} is significantly correlated with defect density shows that the level of detail of messages in sequence diagrams is a significant factor that explains defect density in the implementation. This result is interesting as it shows which aspects of sequence diagram modeling are more important with respect to the quality of the implementation. In this study, we have shown that the increase of SD_{msg} , which can be achieved by specifying non-dummy messages (i.e., messages in sequence diagrams that correspond to class methods), labels in return messages, and parameters in messages, might lead to reduced defect density in the implementation.

One key explanation why LoD of messages in sequence diagrams is significantly correlated with defect density is that sequence diagrams are supposed to specify the control-flow and logic of interaction between components. These aspects are error prone in the implementation. For example, messages in sequence diagrams describe technical details about how several class objects should interact to deliver certain functionality. These details include requested services (method names), required inputs (parameters), and the expected returns. When some of these details are missing from messages, the corresponding implementation might be error-prone.

It is also important to discuss the fact that CD_{aop} is significantly correlated with defect density in the multivariate analysis. Essentially, the presence of significant correlation between CD_{aop} and defect density indicates that an increase in the detailedness of class attributes and operations—more specifically attribute signatures and operation parameters, corresponds to an increase in defect density. However, what is important to note is that the significant correlation is observed *only* when CD_{aop} is combined with complexity and coupling in the multivariate analysis. The univariate analyses do not reveal any significant

correlation between class diagram LoD measures with defect density.

Hence, CD_{aop} may be a useful predictor of defect density—which means including CD_{aop} in a prediction model may improve predictive capability, but it does not confidently explain the underlying relationship between LoD of classes modeled in class diagrams and defect density in the implementation. Nevertheless, we still consider this phenomenon as an open question. One plausible explanation would be that LoD in class diagrams may not have as much significant effect on defect density as LoD in sequence diagrams. This is particularly true because class diagrams are used to define the structure of the implementation. Structural defects are partially detected by compilers and do not surface as run-time errors nor are they found via testing. Therefore, it follows that a poor LoD in class diagrams is less likely to cause defects in the implementation.

6.7.2 Some Remarks for Modeling Practices

The awareness for developing UML models with an appropriate level of quality for the problems at hand has been the subject of many studies. For instance, the risks of defects in UML models and an approach to minimize them have been discussed in [79]. Our case study reported in [95] also reveals why level of detail and completeness in UML models are often sacrificed in practice. Furthermore, a survey reported in [96] captures the opinions of software developers on how level of detail should be applied in UML models: more details should be applied to components that are critical, important, or pertain to concepts that are new to the developers. Apart from these studies, an experimental study reported in [27] suggests how OCL (object constraint language) can help improve understandability of UML models. Similar to the notion of level of detail, the use of OCL is actually aimed at expressing more information (in a formal way) in UML models in order to improve preciseness and reduce ambiguity. A recent study reported in [92] also suggests that LoD in UML models might affect model comprehension.

Considering the results of this study and the aforementioned previous works, we encourage software engineers to pay more attention to the level of detail in UML models. In particular, as this study has shown, we believe that level of detail in behavioral diagrams is a substantial factor for preventing implementation defects. If sequence diagrams are used for modeling system behaviors, then we suggest maintaining the level of detail of messages at a level that assures model comprehension. For example, based on the results of this study we encourage specifying crucial message parameters, specifying important return values in return messages, and avoiding the use of dummy messages.

Nevertheless, we need to emphasize the importance of the question "How much modeling is enough?" Related to LoD in UML models, ideally we should be aware of how far we can put details in UML models to be cost effective. This is particularly true because after a certain point, increasing the level of detail in UML models might not deliver substantial quality improvement. However, determining the break-even point of modeling is still an open question and definitely not a trivial task. Hence, we suggest software designers approach cost effectiveness from a different angle: *where* to focus level of detail such that we can benefit from its application the most. For example, rather than applying similar level of detail on all system parts, designers can prioritize on critical and complex system parts

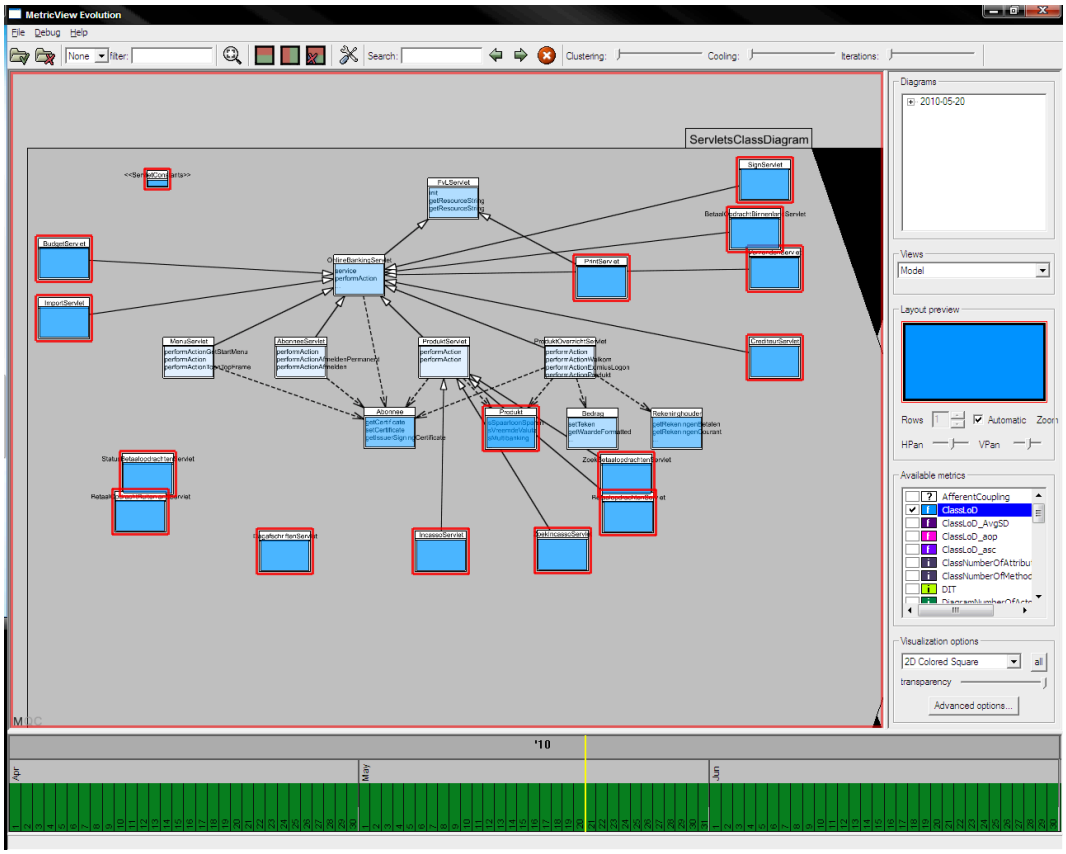


Figure 6.11: The MetricView tool showing the LoD profile of classes in a class diagram. Classes with low LoD are marked with red borders. The color represents the value of an LoD metric. The stronger the color, the lower is the metric value.

(components) so that defects in these parts, which are generally costly, can be prevented or fixed early enough.

To make LoD assessments practical, we provide a tool support (see the discussion in [126]), which is implemented as an extension of the MetricView tool [3]. The tool can assess LoD of UML models that are created using IBM Rational Software Architect and IBM Rational XDE. Additionally, the tool takes XMI files as input. With this extension, the MetricView tool now can calculate LoD metrics of class- and sequence diagrams, and highlight classes and sequence diagrams with poor LoD. Figure 6.11 shows the MetricsView tool displaying the LoD profiles of classes in a class diagram.

6.7.3 Threats to Validity

Internal Validity We identify three threats to the internal validity of this study. The first threat concerns the quality of the UML model (syntactic and semantic) and its effects on defect density. Concerning the syntactic quality, the tool used (Rational XDE) assures that the diagrams comply with the UML 1.4 standard. On the other hand, analyzing the semantic quality (validity) of the diagrams was impossible for us to do because it requires intimate knowledge of the requirements and application domain. Additionally, no engineer from the project was able to aid in this respect. Note, however, that we carefully analyzed the reported bugs. As such, if the UML diagrams were incorrect, the problems would have been reported and then corrected—similar to what we have seen with other defects that were caused by incorrect screen design. Nevertheless, our observations show that none of the sampled defects are caused by incorrect specifications in the UML model. We believe that the semantic quality of the UML model never caused problems because the quality of the diagrams has been checked through two major reviews: 1) design review by designers in the Netherlands; and 2) intake review by developers in India. Only after going through these reviews could the UML model be transferred into the implementation stage.

The second threat is related to different treatments in implementing or verifying different classes. For example, it is logical to think that developers implement or test complex classes more carefully than simpler classes. As a result, the complexity and the amount of testing effort of a class might have a significant influence on its defect density. Related to this issue, we acknowledge the possibility of treating complex and critical components more carefully both during construction and testing. Therefore, in the analysis we also accounted for class complexity so that potential effects of class complexity on defect density (e.g., higher complexity leads to lower defect density) is balanced out.

The last threat concerns inconsistency amongst researchers in analyzing defects. As there were three persons involved in assessing the causes of defects and calculating metrics, individual judgments of the researchers may not always be consistent. This inconsistency could affect the amount of faulty classes and their metrics, and might subsequently lead to inconsistent results. Nevertheless, we have minimized this threat by establishing guidelines, performing joint analysis for a number of cases (to assure a uniform way of working), and by engaging in discussions in cases of doubt.

External Validity The following factors should be considered as threats to the external validity of this study. First, we acknowledge the fact that using only one case study may limit the generalizability of the results of this study. However, we believe that reporting these early findings is necessary as it serves as an encouragement for other researchers to replicate our study using different case studies. Note that later we managed to obtain additional case studies and performed the same analysis. Nevertheless, in the two case studies we did not observe similar results. One important factor that explains the differences is that the additional case studies used UML in a different way. In Chapter 8, we shall discuss this issue further.

The second threat is concerned with the selection of faulty classes in the analysis. Resource constraint has forced us to sample only 28 percent of the 566 findings that are

traceable to the corrected source files, and nearly 15 percent of the traceable findings are useful for further analysis. Furthermore, these 15 percent findings correspond to 122 faulty Java classes, of which 32 classes are modeled in either class or sequence diagrams. The small number of modeled faulty classes used in the analysis might raise an issue concerning the validity of the results. Nevertheless, to verify the consistency of the results we have performed additional analyses in which we use all 566 findings. Results show that the results are consistent, and thus we are assured that the results based on the sampled findings are generalizable to a larger population of faulty classes in the studied case.

6.8 Conclusion and Future Work

In this chapter, we study how the quality of a UML model correlates with the quality of the resulting implementation. To this end, we propose LoD (level of detail) as a measure of the quality of UML models and use defect density as a measure of the quality of the implementation. We define four LoD measures based on class- and sequence- diagrams. Having applied the LoD measures on a substantial industrial Java system, we found a significant correlation between LoD of messages in sequence diagrams and the defect density of the associated implementation classes. The negative nature of the correlation suggests that classes modeled in sequence diagrams with high LoD (i.e., high message detailedness) tend to have a lower defect density in the implementation.

Additionally, we have found that the level of detail in UML sequence diagrams that are used during software maintenance has no significant effect on defect-fixing effort. Similar results may be observed in projects in which most of the engineers remain the same during the project life. In such projects, the engineers who perform defect fixing might have been very well informed about details (both implicit and explicit) in the sequence diagrams based on their exposure to them during the analysis, design, and implementation phases. As such, problems related to LoD in the sequence diagrams may have been clarified earlier, and thus they do not affect the defect fixing activities substantially.

We identify two essential contributions of this study. First, this study proposes practical measures to quantify level of detail in UML models. Second, this study suggests practices to manage LoD in UML models that might reduce incorrect implementation of models.

We realize that more work still needs to be done to assess the usability of the LoD measures defined in this study. Replications of this work using more industrial projects are needed. Through replications we expect to gain more insights about the usefulness of the LoD measures as quality indicators. Furthermore, more research is needed to investigate other design metrics collectible from UML designs and their relations with the quality of the implementation.