UDC 681.3.06

Effectiveness of the User Interface Driven System Design Using UML

Stevan Mrdalj[1], Joseph Scazzero[1], Vladan Jovanovic[2]

[1] Eastern Michigan University
Department of Computer Information Systems
412 Owen, Ypsilanti, MI 48197, USA
(stevan.mrdalj, joseph.scazzero)@emich.edu
[2] Georgia Southern University
Department of Computer Sciences
MPP ADD 3326, Statesboro, GA 30460, USA
vladan@georgiasouthern.edu

Abstract. The majority of research regarding the effectiveness of object-oriented analysis and design (OOAD) is focused on a comparison of object-oriented to traditional approaches that highlights their relative strengths and weaknesses. There has been less focus on improving OOAD on its own. The standardization of the Unified Modeling Language (UML) creates an opportunity to focus on improving the methods of developing UML diagrams. Design quality can be a litmus test for overall system quality. Practice has shown that designing user interfaces before domain modeling can be used on a systematic basis to derive other UML diagrams for a large class of interactive information systems. This empirical study analyzed 43 OOAD projects to determine the effectiveness of the user interface driven system design (UIDD) by calculating defect densities for four UML diagrams. The study was performed on three levels: individual type of defect, type of diagram and entire project. Empirical results show that the UIDD consistently produced very low defect densities on all three levels for projects that varied widely with respect to application area, information system type, team experience, and size.

1. Introduction

An important area in information systems research is the evaluation of the effectiveness of object-oriented analysis and design (OOAD) methodologies. If an information system has a flawed design, it is likely to have other quality problems. Conversely, an effective design is likely to enhance the quality of the system's other parts. To date, the majority of the research is focused on a comparison of object-oriented to traditional methodologies that highlights their relative strengths and weaknesses. The standardization of the Unified Modeling Language (UML) eliminates disputes over using different variations of diagrams for the same purpose. It also creates an opportunity to evaluate the effectiveness of the different approaches used to create a certain type of UML diagram.

Most of the object-oriented design approaches/methodologies [10, 12, 16] emphasize system decomposition into objects in the early analysis phase. The noun phrase [2], conceptual class category list [13] and analysis patterns [9] are commonly used techniques in the process of domain modeling. All three techniques are used to make a list of candidate classes. Larman [13] states, “It is better to over specify a domain model … than to under specify it” as the usual guidelines in identifying conceptual classes. Approaches like this require considerable effort to identify all the possible classes during the early elaboration phase of the project and to verify if they are all needed or if all needed classes are identified. Consequently, all developed diagrams need to be refined using candidate classes which are modified or eliminated. This verification concludes at the design phase using detailed user interface (UI) design.

In order to eliminate the considerable refinement caused by candidate classes, a user interface driven design (UIDD) approach [12] suggests performing a detailed UI design before domain modeling and to use it as a basis to identify classes and to develop all other diagrams. The user interface driven design is certainly not new. Indeed, savvy developers [8, 10, 11, 14, 17] analyze the UI and convert it into data and code. This way, the initial domain model will contain only the necessary classes and interactions among them. Consequently, this approach is well suited for a large class of interactive information systems that can be alternatively categorized as systems with substantial externally visible behavior. An initial domain model can be further refined to reflect various nonfunctional requirements such as performance, expandability, and maintainability. UIDD uses UML diagrams [2] since they are the de facto industry-standard modeling notation for object-oriented development.

The objective of this paper is to empirically evaluate the effectiveness of the UIDD approach in helping analysts convert user requirements into an object-oriented specification of those requirements and to serve as a foundation for improving OOAD using UML. We wanted to evaluate the effectiveness of this approach by determining how correctly and how consistently it produces UML diagrams. We also wanted to evaluate how invariant the results are with respect to the application area, information system type, team experience and size of a project. To answer these questions, we conducted an in-depth evaluation of diagrams produced using the UIDD. The study analyzed projects developed by senior-level graduate students taking a required systems analysis and design class over a period of seven semesters.

Since there has been no systematic research on how to measure the effectiveness of creating UML diagrams, we established a set of formally defined defects for each type of diagram that could violate a well-specified system design [3, 6, 18] and we used defect density [7] as a measurement tool. Defect density, which measures the correctness of derived diagrams, is an indicator of the effectiveness of the UIDD. Statistical tests of hypotheses were performed on defect densities for the entire project, each type of diagram and each type of defect within a diagram. We recognize that the correctness of derived diagrams can only provide a partial answer for effective OOAD. However, our objective is to test the benefit of the UIDD approach in developing correct UML diagrams.

This paper is organized as follows. Section 2 reviews a UIDD approach and its steps used in this study. Section 3 presents an approach for measuring diagram derivation effectiveness. Section 4 describes our empirical evaluation methodology. Section 5 contains the data analysis and findings of our research. Section 6, addresses some of the potential limitations of this study that are related to the research methodology. Finally, the paper concludes with a discussion of the results.

2. User Interface Driven System Design Steps

The goal of UIDD is to minimize the possibility of overlooking UI requirements by developing them early in the design process and by using them as a basis for developing all other components of the system model.

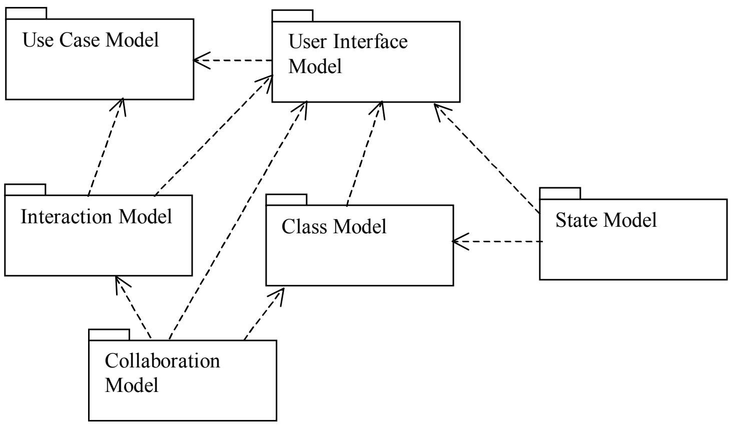

Fig. 1. System Model Dependencies

As Figure 1 shows, a UIDD process can be performed in different ways by following the dependencies among packages. For the purpose of this evaluation, we will assume that the Use Case Model and User Interface Model are given as an initial user requirement from which we need to develop a design model. The principles and the process of developing use case diagrams, UI design as well as a detailed description of the UIDD can be found, for example, in [15]. The design process described in this paper uses the following steps.

2.1. Deriving Class Diagrams from User Interfaces

The subjects were instructed to use the following rules in deriving class diagrams from the UI. The existence of a field on a dialog/Web page/report meant that the data must either be an attribute of some object, the result of some operation on an object or series of objects, or be calculated from the attributes of an object(s). Existence of the data about different objects on the same user interface means that those objects are related to each other and results in an association between classes in the class diagram. Initial multiplicities for such associations might be detected by the occurrences of the related data (objects). The use of UI forms and reports to capture requirements for database design are also demonstrated in [11].

2.2. Deriving Sequence Diagrams from the User Interface

The subjects were instructed to use sequence diagrams to formally describe UI navigation by applying the following rules. Each form of interaction with a use case requires an appropriate representation in the sequence diagram. In other words, each interface prototype is associated with a central view class and each window/dialog corresponds to a view class. At the same time, the existence of each form means that there must be a view class in the view class model. We use the view class/model name to avoid confusion with interface classes from UML. Menu options or buttons on screens typically trigger events sent either to a subsequent screen/dialog or to the application. Omission of any of these classes can cause omission of a large number of required operations for these classes.

The screen/dialog flows can lead to the definition of the interactions in the sequence diagram. For each screen/dialog, all events generated through its buttons and menus need to be captured by the appropriate messages/events in a sequence diagram.

If a sequence diagram covers the interaction among business classes, then each business class in a sequence diagram has to exist in the class diagram. The fact that the object of one class sends a message to the object of another class means that there needs to be an association between these classes. The exception to this rule is the procedural relationship for collaboration diagrams as described in the next section.

2.3. Deriving Collaboration Diagrams from the User Interface

Since a collaboration diagram emphasizes the organization of the objects that participate in an interaction, the subjects were instructed to use the following rules in deriving collaboration diagrams from the UI. Each <<business>> object from a collaboration diagram has a corresponding class in a class diagram. Each link between two objects, that is not otherwise stereotyped, has to correspond to an existing association in a class diagram. The role names for those links and associations need to be the same. A possible exception to this rule is the case of "procedural relationships" in which there will be no explicit association between these classes.

2.4. Deriving Statechart Diagrams from the User Interface

A statechart diagram can be attached to a class, a use case or even an entire system to model their dynamic aspects. For the most part, it is used for modeling the behavior of reactive objects–whose behavior is best characterized by their response to events dispatched from outside their context. Therefore, such business objects and almost all view objects may have associated statechart diagrams. In their projects, subjects were instructed to use statecharts to formally represent user interaction with each form in the given UI. They were also instructed to use statechart diagrams for the most representative UI. In deriving statechart diagrams from the UI, each independent event like data entry or button click needs to be appropriately modeled as a transition in a statechart. The subjects where encouraged to use composite states to model non-sequential transitions. All operations that are listed as part of a transition specification must also exist in the class diagram for the appropriate class. The same principle applies for all operations listed as actions or activities for the states. At the same time, role names used in a state chart must correspond to the appropriate role names in a class diagram.

3. Measuring Diagram Derivation Effectiveness

3.1. Defect Density

A de facto standard measure of software quality is defect density. Similarly, the effectiveness or ineffectiveness of UIDD can be determined by the number of defects made in deriving the diagrams. Thus, defect density (DD) will be used as a measure of diagram derivation quality where:

$$DD = \frac{number \quad of \quad known \quad defects}{number \quad of \quad use \quad cases} \tag{1}$$

Since we are evaluating the effectiveness of an object-oriented approach, in our study we used the number of use cases as a relative measure of the project size instead of a traditionally used measure such as KLOC [7]. We computed three types of defect densities: 1) the defect density made within the entire project (DD_P); 2) the defect density made in deriving a particular type of diagram (DD_D) for all projects; and 3) the defect density for a certain type of defect within a diagram (DD_C) for all projects.

3.2. Types of Defects

We used empirically found defects to determine the correctness of each type of diagram. Table 1 lists the types of defects found by type of diagram.

Table 1. Types of Defects for Different Types of Diagrams

	Type of Diagram			
	Class	**Sequence**	**Collaboration**	**Statechart**
Type of Defect	Missing a class	Missing an object	Missing an object	Missing a state
	Missing an attribute	Missing an operation	Missing a link	Missing a transition
	Missing an association	Missing a control structure	Missing a message	Missing a start/end
	Needless class	Needless object	Needless object	Needless state
	Needless attribute	Incorrect object name	Incorrect object name	Needless start/end state
	Incorrect multiplicities	Incorrect type of the control structure	Wrong order of messages	Wrong superstate name
	Wrong class name	Incorrect order of the control structure	Needless link	Wrong state
	Wrong association name	Needless message	Incorrect message	Incorrect superstate
	Needless associations	Wrong message		Needless transition
	Incorrect aggregation			Wrong transition name

	Incorrect inheritance			Missing superstate
	Incorrect attribute name			
	Missing inheritance			
	Missing aggregation			

A diagram is correct when the above defects are not found for that type of diagram. It is important to note that an analyst can develop multiple diagrams of equal correctness. The correct design model does not assume that there is one best way to model the system. In this paper, we are not interested in comparing "equivalent" models but rather the effectiveness of deriving diagrams using UIDD.

4. Evaluation Methodology

4.1. Evaluation Participants

The study analyzed 43 projects developed by a total of 211 senior-level graduate students enrolled in a Master's of Science in Information Systems program. The 43 projects were collected over a period of seven semesters. Participants self-divided into teams that had an average size of 4.9 students. Each team had to find a local business for which they had to design an information system. There were no multiple projects for the same company. No particular order was used to assign subjects to teams and teams randomly selected their project application area. The duration of the project was the entire semester. All teams used the same CASE tool with minor differences due to different versions. While participants were familiar with UML and the system domain, they could not be considered experts in either area.

4.2. Evaluation Procedure

During project development, all diagrams were cross-checked at each delivery point of the model. Students were allowed to revise their diagrams until their final project submission at the end of the semester. At the end of each semester, each project was evaluated by an instructor, who has extensive experience in OOAD and UML and has numerous publications in this area. To increase validity, a second evaluator who is well trained in UML independently scored defects of all projects. This evaluation consisted of counting the number of participating concepts in a given diagram and the number of defects. Seven projects were used in a pilot evaluation. Minor changes were made to the evaluation instrument based on the feedback from the pilot study.

To determine if any UML diagram had a defect, an external view of the UI was used as a reference point to evaluate the diagrams. The emphasis was given to the ability to "evaluate by scenario" [4]. Each defect was appropriately marked and reviewed by both evaluators. Any differences between evaluators with respect to a defect, which occurred infrequently, were discussed until a consensus was reached. It should be noted that the first evaluations were conducted without knowledge that this study would be conducted later.

4.3. Evaluation Instruments

In order to tabulate defects, scoring tables were developed for each of the diagrams. Figure 2 shows the layout of the scoring table for class diagrams. The scoring tables for other diagrams have a similar format with columns named for the concepts associated with that type of diagram. These tables were used by second evaluator.

Class				Attribute				Associ ation				Multip licity		Inherita nce			Aggrega tion/ Compos ition		
Number	Wrong name	Missing	Needless	Number	Wrong name	Missing	Needless	Number	Wrong name	Missing	Needless	Number	Incorrect	Number	Missing	Incorrect	Number	Missing	Incorrect
2				3			1	2				4		1					
4		1		8				11				22					1		
4				9				10				20	1	1		1			
2				7		1		1				2							

Fig. 2. Scoring Table for Class Diagram

5. Data Analysis and Findings

As mentioned earlier, the students that participated in this case study handed in a total of 43 final projects. Descriptive statistics were obtained for these 43 projects on their characteristics and their defect densities. The projects averaged 11.5 use cases, 6.0 actors, 16.3 interactions, 1.3 includes, 1.1 extends, and 0.4 inheritances. The number of use cases per project showed an almost uniform distribution, ranging from 6 to 19 use cases per project.

The projects had a total of 557 class diagrams, resulting in an average of 13.0 diagrams per project. The class diagrams had an average of 8.8 classes and 13.6 associations. There were a total of 483 sequence diagrams, averaging 11.2 diagrams per project. Each sequence diagram had on average 4.1 objects and 17.3 operations/messages. We examined a total of 108 collaboration diagrams, an average of 2.5 diagrams per project. The smaller number of collaboration diagrams relative to the number of sequence diagrams is a result of the subjects' selection preference among comparable tools. The collaboration diagrams averaged 4.7 objects, 4.8 links, and 7.0 operations/messages. Subjects developed a total of 68 statechart diagrams, an average of 1.6 diagrams per project, where each statechart diagram averaged 7.1 states and 11.2 transitions/events.

The most important indication of the effectiveness of the UIDD is how correct the produced diagrams are. The correctness of the diagrams is demonstrated by the low defect density for each type of diagram (DD_D) as shown by the boxplots in Figure 3. Boxplots are used to show the distributional characteristics of DD_D. The line in the box is drawn at the median while the bottom of the box is at the first quartile (25th percentile) and the top of the box is at the third quartile (75th percentile). Points outside the lines from the box are considered outliers and are indicated by asterisks.

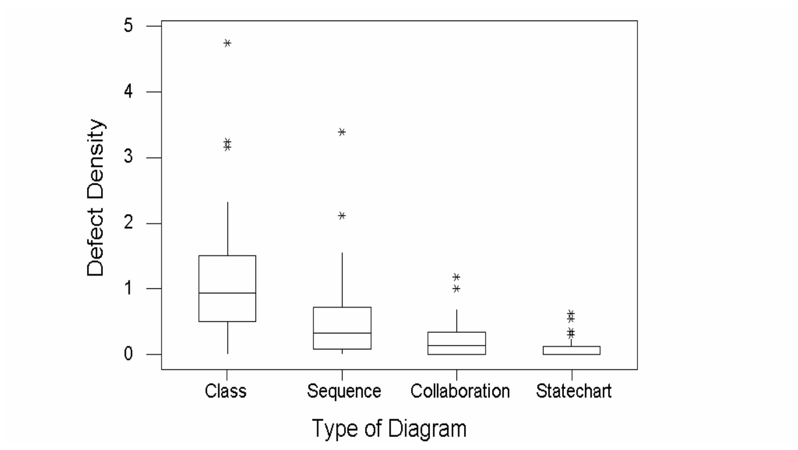

Fig. 3. Boxplots of Defect Density Values by Type of Diagram

It is extremely difficult to asses the goodness of the obtained results since the authors are not aware of any similar study and therefore are unable to perform real comparisons. For illustration purposes, we used the quantitative targets for managing US defense projects[1] [5], according to which an effective design method should produce a defect density that is less than 4, whereas an ineffective method produces a defect density that is greater than 7. As seen in this figure, all diagrams are well within the effective level even though class diagram had one outlier project that was outside the effective range but still below the ineffective level.

Another indication of the effectiveness of the UIDD is the low defect densities for the individual types of defects (DD_C) within each diagram. The average DD_C found in class diagrams, sequence diagrams, collaboration diagrams, and statechart diagrams are given in the Tables 2 through 5 below. The types of defects are ordered in the decreasing order of their average defect densities. It should be noted that these low defect densities were achieved for projects that varied widely with respect to application area, system types, and size.

Table 2. Average Defect Density by Type of Defect in Class Diagrams

Type of Defect	Average DD_C
Incorrect multiplicities	0.528
Needless associations	0.122
Missing an association	0.114
Wrong class name	0.078

[1] It is important to point out that US defense department projects differ considerably in nature from the projects used in this study and that they tend to be extremely complex.

Wrong association name	0.066
Needless class	0.060
Missing a class	0.056
Needless attribute	0.037
Missing an attribute	0.021
Incorrect inheritance	0.021
Incorrect aggregation	0.014
Incorrect attribute name	0.003
Missing aggregation	0.003
Missing inheritance	0.002

Table 3. Average Defect Density by Type of Defect in Sequence Diagrams

Type of Defect	Average DD_C
Wrong message	0.209
Needless message	0.116
Missing an operation	0.090
Needless object	0.035
Missing an object	0.020
Incorrect order of the control structure	0.019
Incorrect object name	0.018
Missing a control structure	0.010
Incorrect type of the control structure	0.003

Table 4. Average Defect Density by Type of Defect in Collaboration Diagrams

Type of Defect	Average DD_C
Incorrect message	0.045
Wrong order of messages	0.040
Missing a link	0.034
Missing an object	0.028
Missing a message	0.027
Needless object	0.015
Incorrect object name	0.010
Needless link	0.005

Table 5. Average Defect Density by Type of Defect in Statechart Diagrams

Type of Defect	Average DD_C
Wrong transition name	0.019
Needless transition	0.014
Incorrect superstate	0.012
Missing a transition	0.010
Wrong state	0.010
Needless state	0.007
Missing a state	0.004
Needless start/end state	0.003
Missing superstate	0.002
Missing a start/end	0.000
Wrong superstate name	0.000

Such very low average defect densities for all types of defects in the class diagrams seem to support the effectiveness of the UIDD approach. These defect densities certainly compare favorably to the average grades, ranging from 1.7 to 2.2 on the scale of 0 (lowest) to 4 (highest), obtained through reverse engineering for 42 case studies involving database design and modeling [1].

5.1. Consistency of the UIDD with Respect to the Application Area and System Type

As we mentioned before, we wanted to determine if the UIDD consistently produced the low defect densities shown above regardless of application area, information system type or implementation environment. UIDD can be considered consistent for a diagram or defect if the average defect density for that diagram or defect was the same over time, i.e., across the seven semesters of the study. Thus, in order to determine if the UIDD is consistent for class diagrams, the following hypothesis was tested:

$H1_0$: The average defect density for class diagrams is the same across semesters.

$H1_1$: $H1_0$ not true.

We used one-way analysis of variance (ANOVA) for testing $H1_0$. Specifically, ANOVA tests for the equality of the population average defect density for class diagrams across semesters. Based on the ANOVA results for class diagrams shown in Table 6, we do not reject $H1_0$ for $\alpha = 0.01$ and conclude that the UIDD produces consistent results for class diagrams. Similarly, we also conclude that the UIDD produces consistent results for sequence, collaboration, and statechart diagrams.

Table 6. ANOVA Results for Testing the Eqality of the Average DD_D Across Semesters

Diagram	**F(6,36)**	**p-value**
Class Diagram	1.04	0.419
Sequence Diagram	1.43	0.230
Collaboration Diagram	2.83	0.023
Statechart Diagram	2.89	0.021

Another way to asses the effectiveness of the UIDD is to consider confidence intervals for the average defect density (DD_D) for each type of diagram. The 95% confidence interval for the average DD_D for class diagrams is between 0.84 and 1.43, for sequence diagrams between 0.30 and 0.70, for collaboration diagrams between 0.12 and 0.29, and for statechart diagrams between 0.04 and 0.13. Thus, we can conclude that the UIDD consistently produces very low average defect densities for all four types of diagrams regardless of application area, information system type or implementation environment.

Next, we wanted to determine if the UIDD is consistent with respect to the types of defects associated with each type of diagram presented in Tables 2 through 5. Table 7 shows the ANOVA results for testing the equality of the average DD_C across semesters by type of defect in class diagrams shown in Table 2. For example, for the first type of defect in Table 7, the hypothesis is:

$H2_0$: The average defect density of missing classes in class diagrams is the same across semesters.

$H2_1$: $H2_0$ not true.

Table 7. ANOVA Results for Testing the Eqality of the Average DD_C Across Semesters by Type of Defect in Class Diagrams

Type of Defect	F(6, 36)	p-value
Missing a class	0.69	0.657
Missing an attribute	2.30	0.056
Missing an association	0.29	0.939
Needless class	0.92	0.492
Needless attribute	1.84	0.118
Incorrect multiplicities	2.91	0.020
Wrong class name	1.46	0.219
Wrong association name	0.67	0.677
Needless associations	1.49	0.211
Incorrect aggregation	0.60	0.729
Incorrect inheritance	0.77	0.601
Incorrect attribute name	0.39	0.880
Missing inheritance	0.59	0.734
Missing aggregation	1.03	0.420

Based on $F = 0.69$ with a p-value of 0.657, $H2_0$ would not be rejected for $\alpha = 0.01$. Thus, we conclude that the UIDD consistently produces low average defect density with respect to missing classes in class diagrams. Similarly, consistency was found to exist for the other types of defects in class diagrams listed in Table 7. This implies that the UIDD produces consistent average DD_C for all types of defects in class diagrams. However, an in-depth examination of the data showed that for α = 0.05, there is a significant difference across semesters for the incorrect derivation of multiplicities. This indicates that a more comprehensive study is needed to improve directions (procedures) for deriving multiplicities in the UIDD.

The equality of the average DD_C across semesters was also tested for all types of defects in sequence diagrams. Table 8 shows the ANOVA results for these tests.

Table 8. ANOVA Results for Testing the Eqality of the Average DD_C Across Semesters by Type of Defect in Sequence Diagrams

Type of Defect	F (6, 36)	p-value
Missing an object	1.21	0.325
Missing an operation	2.57	0.036
Missing a control structure	1.21	0.322
Needless object	1.92	0.105
Incorrect object name	0.97	0.461
Incorrect type of the control structure	2.79	0.025
Incorrect order of the control structure	2.23	0.038
Needless message	4.58	0.001*
Wrong message	1.27	0.294

*Significant at α = 0.01

For α = 0.01, we can conclude that consistency exists for each type of defect in sequence diagrams except for the deriving of needless messages. The significant difference between averages for this defect was primarily due to a higher average of "needless messages" in the first two semesters. This was corrected by improved derivation procedure in subsequent semesters. Thus, except for this one type of defect, the average number of defects does not change across semesters for sequence diagrams. Similarly, for α = 0.01, the UIDD was also consistent for each type of defect in collaboration and statechart diagrams.

5.2. Invariance of the UIDD with respect to the size of the project

The size of projects in this study ranged from 6 to 19 use cases per project. We also wanted to determine if the UIDD is invariant with respect to project size. The UIDD can be considered invariant for a diagram or defect if no relationship exists between defect densities for that diagram or defect and size of project where size is measured by the number of use cases in the project. In other words, invariance implies that the defect density for a diagram or defect does not change as the size of the project increases. In order to determine if the UIDD is invariant for class diagrams, the following hypothesis was tested:

$H3_0$: No linear relationship exists between defect density for class diagrams and number of use cases

$H3_1$: A linear relationship exists between defect density for class diagrams and number of use cases

The sample Pearson product moment correlation coefficient r, which measures the strength of the linear relationship between two variables, was used to test the above hypothesis. It should be noted that only tests for linear relationships were necessary since an examination of the scatter plots showed that only linear relationships were present between the defect density for a particular diagram (DD_D) and the number of use cases. Based on the class diagram p-value in Table 9, we would not reject $H3_0$ for $\alpha = 0.01$ and conclude that the UIDD is invariant for class diagrams or that the defect density for class diagrams does not change as the size of the project increases. Similarly, we also found that the defect density for sequence, collaboration, and statechart diagrams does not change as the size of the project increases.

Table 9. ANOVA Results for Testing the Relationship Between DD_D and Numbr of Use Cases

Type of Diagram	r	p-value
Class Diagram	-0.36	0.018
Sequence Diagram	0.17	0.275
Collaboration Diagram	0.23	0.140
Statechart Diagram	0.054	0.732

Similar tests of hypothesis showed that the UIDD was invariant for each type of defect found in these diagrams, i.e., no linear relationship was found between the defect density for each type of defect, DD_C, and the number of use cases. Thus, we can conclude that the derivation of all type of diagrams using the UIDD is invariant with respect to project size.

5.3. Overall Project Level Effectiveness

A measure of overall project level effectiveness is the defect density for the entire project, DD_P, which can be defined as the total number of defects found in all diagrams divided by the number of use cases in the project.

Equivalently, the project defect density is equal to the sum of the individual densities discussed earlier, i.e., the sum of defect densities for the class diagram, sequence diagram, collaboration diagram, and statechart diagram.

The overall effectiveness of the derivation of all diagrams using UIDD within a project is illustrated by the boxplot in Figure 4.

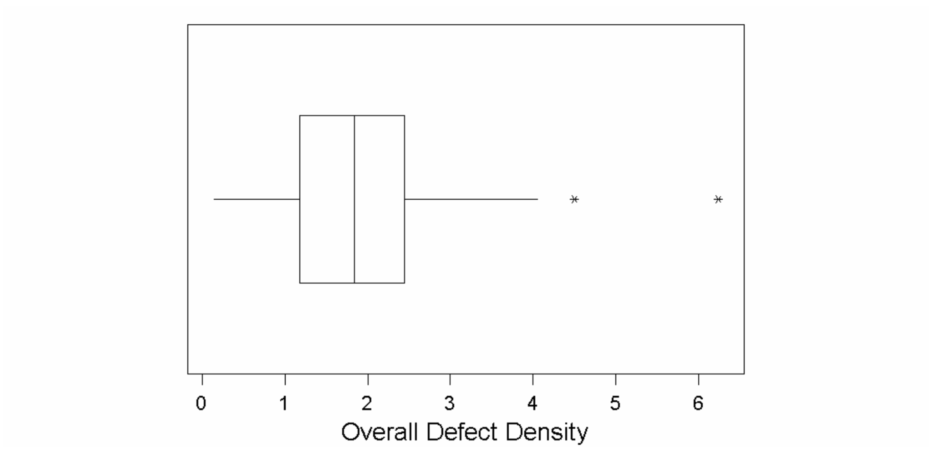

Fig. 4. Boxplot of DD_P values

The average project defect density was the same across all semesters (F(6,36) = 1.19, p-value = 0.334) so the UIDD was consistent for all projects with respect to application area, information system type, and implementation environment. The 95% confidence interval for the average project defect density is between 1.56 and 2.29. The project defect density did not change as the size of the project increased (r = -0.13, p-value = 0.400) so the UIDD is also invariant with respect to the DD_P and project size.

6. Limitations

The main purpose of this study is to evaluate the effectiveness of the UIDD. This study would be incomplete without an examination of the limitations of its research method.

In calculating defect densities, we used the defects found in the design phase of the SDLC versus the defects found in an operational system. Since the absence of defects in the system design does not guarantee correct system operation, the operational defect densities might differ from our results.

The scope of this study may limit the generalization of results with respect to the complexity of the systems. The limitation of the presented results may come from the fact that use cases vary widely in their complexity, which we did not explicitly address in this study. One possible indication of the use case complexity is the complexity of the associated sequence diagram. Nevertheless, the results suggest that there is no significant difference in the defect density for the use cases whose associated sequence diagrams had from 1 to 22 classes, from 2 to 101 messages and from 1 to 48 control structures. Despite these results, which suggest UIDD's scalability, further investigation is necessary to validate these results for much larger systems that have varying degrees of complexity for their use cases.

This study did not measure inter-rater reliability, since there were very few disagreements with respect to defects during the pilot evaluation. One area of disagreement was in the categorization of the defects, which was resolved during the pilot evaluation process. Thus, because of the extremely low rate of disagreements between evaluators, we decided not to perform an inter-rater reliability study for the full project.

The use of students in a research study may limit the potential applicability of its results. In our opinion, considering the nature of the UIDD process, it can be expected that professionals with experience in OOAD and UML should produce even fewer defects and therefore, that would not undermine the conclusions of this study.

7. Conclusions

The system design quality is undeniably a good indicator of application quality. The purpose of this paper is to examine the effectiveness of the user interface driven design (UIDD) as a basis for developing class, sequence, collaboration and state diagrams. This paper used defect density (number of known defects per use case) to measure the effectiveness of the UIDD in diagram derivation. We analyzed empirical project data in this paper on three levels: individual type of defect, type of diagram and entire project. The very low defect densities for individual defects, individual diagrams and entire projects indicate that the UIDD produces a good quality design.

Based on ANOVA tests, the UIDD consistently produces low defect densities on all levels (project, diagram and individual defect) regardless of the application area, information system type and implementation environment. Lastly, based on correlation tests, we also conclude that the UIDD is invariant with respect to the tested project sizes for all three levels of study.

We believe that our results have strong implications for educators/practitioners involved in selecting a system development process. We consider the main benefits of the UIDD to be its wide applicability to various types of businesses (no major differences among types of projects) and its wide applicability to a variety of UML diagrams (no major differences in correctness among types of diagrams).

The limitations of this study point to directions in which the research presented here can be extended by future investigations. First, similar studies can be conducted for other object-oriented design approaches. The effectiveness of these approaches can then be compared with the UIDD. Second, further investigation is necessary to consider the complexity of a project's use cases. Third, performing a similar study with practitioners would be extremely valuable since their effectiveness with UIDD may differ from that found in an academic environment.

References

1. Blaha, M.: A Copper Bullet for Software Quality Improvement. IEEE Computer, Vol. 37, No. 2, 21-25. (2004)
2. Booch, G. Rumbaugh, J. and Jacobson, I. The Unified Modeling Language User Guide, Addison-Wesley, Reading, MA, USA (1998)
3. Chidamber, S.R., and Kemerer, C.F.: A Metrics Suite for Object-Oriented Design. IEEE Transaction on Software Engineering Vol.20, No.6, 476-493. (1994)
4. Code, P. and Yourdon, E.: Object-oriented analysis, 2nd ed. Yourdon Press, Englewood Cliffs, NJ, USA (1991)
5. Department of the Navy (US): Software Program Managers Network. NetFocus, No. 207, January. (1995)
6. Dromney, R.G.: Cornering the Chimera. IEEE Software Vol.13, No.1, 33-43. (1996)
7. Fenton, N.F. and Pfleeger, S.L.: Software Metrics: A Rigorous & Practical Approach, 2nd ed. PSW, London, UK. (1996)
8. Fertuck, L.: System Analysis and Design: With Modern Methods. Wm. C. Brown Comm., Dubuque, Iowa, USA. (1995)
9. Fowler, M.: Analysis Patterns: Reusable Object models. Addison-Wesley, Reading, MA, USA. (1997)
10. Gossain, S.: Tracking Requirements in Object Development: Part II. Report on Object Analysis and Design, Vol.2, No.3, 18-19,55. (1995)
11. Jovanovic, V. and Mrdalj, S." Three-Layered Approach to the Analysis of Forms and Transactions. In Proceedings of the IAMM Conference, Dallas, TX, USA, 141-148. (1990)
12. Kruchten, P.: Rational Unified Process, 2nd ed., Addison-Wesley, Reading, MA, USA. (2000)
13. Larman, C.: Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design, Prentice Hall, Upper Saddle River, NJ, USA. (1988)
14. Lee, H., Yoo, C.: A Form Driven Object-Oriented Reverse Engineering Methodology. Information Systems, Vol. 25, No. 3, 235-259. (2000)

15. Mrdalj, S. and Jovanovic, V.: User Interface Driven System Design. Issues in Information Systems Vol.3, No.1, 441-774. (2002)
16. Rosenberg D.: Use Case Driven Object Modeling with UML. Addison-Wesley, Reading, MA, USA. (1999)
17. Van Hartman M., editor: Object Modeling and User Interface Design, Addison-Wesley, Reading, MA, USA. (2001)
18. Wand, Y. and Weber, R.: A Model of Systems Decomposition. In Proceedings of the Tenth International Conference on Information Systems, Boston, MA, USA, 42-51. (1989)

Stevan Mrdalj is a Full Professor of Computer Information Systems at Eastern Michigan University, USA. He received his Ph.D. in information systems from the University of Belgrade, Serbia and Montenegro. He is a member of ACM, AIS and IEEE. His research interests are in the areas of framework based enterprise architecture, user interface driven system development and design patterns. He has written numerous articles in the area of object-oriented system development.

Joseph A. Scazzero is an Associate Professor of Decision Sciences in the Department of Commuter Information Systems at Eastern Michigan University, USA. His research interests include experimental design, quality control, and software engineering. His publications have appeared in Communications in Statistics, Information & Management, Journal of Statistical Planning and Inference, International Journal of Quality and Reliability Management, Total Quality Management, and Information Systems Management Journal. He has also has worked in private industry and the federal government as a statistician. Professor Scazzero received a Ph.D. (Applied Statistics) and M.A. (Statistics) from The Pennsylvania State University, M.S. (Computer Science) from Johns Hopkins University, and A.B. (Mathematics) from Rutgers College.

Vladan Jovanovic is currently a Professor of Computer Sciences at the Georgia Southern University, USA. He obtained a Ph.D. from the University of Belgrade, Serbia and Montenegro in 1982. Dr. Jovanovic is a member of the IEEE, ACM, AIS, and NY Academy of Sciences. He is (co-)author of six books and 28 journal articles. His research interests focus on the information and software systems design and management. His major contributions are in the areas of design methodology and curriculum development in all fields of information technology. He is actively involved in writing IEEE Software Engineering Standards and the establishment of ASQ Software Quality Engineering Certification.