

Is In-Depth Object-Oriented Knowledge Necessary to Develop Quality Robustness Diagrams?

Mohamed El-Attar, Mahmoud O. Elish, Sajjad Mahmood
 Department of Information and Computer Science
 King Fahd University of Petroleum and Minerals
 P.O. Box 5066, Dhahran 31261, Kingdom of Saudi Arabia
 {melattar, elish, smahmood} @kfupm.edu.sa

James Miller
 Department of Electrical and Computer Engineering
 University of Alberta
 Edmonton, Alberta T6G 2V8, Canada
 jm@ece.ualberta.ca

Abstract—PURPOSE: Robustness analysis is a technique that can be performed to help ensure the correctness, completeness and consistency of use case and domain models. Robustness analysis also helps bridge the gap between the analysis and design phases by providing a guided approach to identify a first-guess set of objects that will realize scenarios described in use cases. It is necessary to perform robustness analysis in the early phases of the development lifecycle in order to reap its benefits. In particular, robustness analysis needs to be performed by business analysts during the requirements phase to improve the quality of their models as well as help provide a seamless transition to the design phase. However, a core skill that is required to develop robustness diagrams is knowledge of OO concepts which business analysts normally do not have. To overcome this limitation, business analysts acquire brief knowledge of OO concepts via a small learning curve in order to develop and reap the benefits of creating robustness diagrams. However, is this brief knowledge of OO concepts attained through a small learning curve enough to allow business analysts to develop quality robustness diagrams?

DESIGN: In this paper we present a controlled student-based experiment to empirically evaluate the requirement of in-depth OO knowledge to produce quality robustness diagrams.

FINDINGS: The results show that business analysts can indeed produce quality robustness diagrams without in-depth OO knowledge.

ORIGINALITY: The results of this experiment will aid in embracing the technique of robustness analysis amongst business analysts in order to overall improve the software development process.

Index Terms— Robustness Analysis, Robustness Diagrams, Use Cases, Business Analysts, Controlled Experiment

I. INTRODUCTION

Robustness analysis is a technique which is first introduced by Ivar Jacobson in 1993 [15] which is performed to disambiguate use cases and identify gaps in the domain model. A domain model is essentially a class diagram that represents elements from the real-world domain (the problem domain). There exist several benefits to applying robustness [25]. Firstly, robustness analysis is used to disambiguate and complete a use case description by rewriting it using elements from the domain model. Secondly, robustness analysis is used to deduce candidate classes from use case descriptions in order to complete the corresponding domain model. As a result, the use case and domain models will be consistent and more complete. Thirdly, robustness analysis is used to identify a set of objects that will collaboratively realize the behavior described in use case descriptions.

The deliverable of robustness analysis is a robustness diagram. A robustness diagram models the behavior described by a use case using objects from existing classes in the domain model as well as newly introduced classes. Robustness diagrams are most similar to UML collaboration diagrams but with far less diagrammatic constructs and syntax rules. Upon selecting the most appropriate design, a robustness diagram can be readily evolved into more detailed UML design artifacts such as collaboration, activity or sequence diagrams, whose notational sets subsume that of robustness diagrams. Therefore, robustness analysis and robustness diagrams can be considered as a valuable tool to bridge the gap between the analysis and design phases and can steer development efforts towards developing an end system that more precisely satisfies its requirements.

Due to their relatively simple notational constructs and syntactical rules, robustness diagrams can be produced

early in the development life cycle. Development teams can benefit from this characteristic by using robustness diagrams to obtain early feedback from their customers and avoid costly fixes downstream. Robustness diagrams can also be potentially used to develop more comprehensive sets of acceptance, unit, integration and system tests. Moreover, due to their simplicity, they are easier to comprehend than detailed UML artifacts and thus are very useful for the software maintainers to grasp an overview of the system's behavior without being distracted with the intricate design details.

The literature provides some evidence of the use of robustness analysis in industry. In [26], the authors describe an industry strength software development process named ICONIX, which leverages the benefits of robustness analysis. Robustness analysis is stated to play several essential roles within the ICONIX process [26]. In [1], Aguanno states that robustness diagrams can be used as a modeling technique to develop domain and conceptual models in agile projects. "Agile Modeling" is a practice-based methodology for modeling and documenting software systems, which utilizes robustness analysis and diagrams [2]. In [24], the author presents an approach to better apply robustness analysis in conjunction with the Model-View-Controller architecture. In the research community, Dugerdil and Jossi presented a technique that can reverse engineer the architecture of legacy software systems [9]. Building a robustness diagram is described as a major step in the application of the approach presented in [9]. A number of major companies who develop UML modeling tools have also realized the potential advantages of robustness analysis. To this end, many commercial UML modeling tools have invested in providing support for robustness diagrams within their tools. Such tools include: astahUML [5], ModelMaker [23], Visual Paradigm for UML [31], Enterprise Architect [10] and MagicDraw UML [22].

The abovementioned benefits of robustness analysis and robustness diagrams will not be reaped unless it is performed early in the software development cycle. Therefore, there is a need for business analysts to embrace robustness diagrams before designers commence with their detailed design efforts. Although developing robustness diagrams does not require in-depth knowledge of OO concepts as is the case with UML collaboration diagrams, nevertheless some level of knowledge of the OO concepts is required since ultimately a robustness diagram is essentially a depiction of objects collaborating with each other in order to realize scenarios described in use cases. However, according to BABOK® (Business Analysts Body of Knowledge) [14], knowledge of OO concepts is not a requisite skill to obtain professional accreditation, not even for the most advanced level of accreditation. While business analysts may possess other skills which may (positively or negatively) affect their ability to produce quality robustness diagrams, it can be argued that knowledge of OO concepts is a requisite core skill, if not the most important skill. To overcome this limitation, business analysts acquire a brief amount of OO knowledge with a short learning curve, perhaps

through a short course or a workshop. However, there lacks evidence that the brief knowledge acquired by business analysts is adequate to develop quality robustness diagrams. It is thus necessary to empirically evaluate whether in-depth knowledge of OO concepts is necessary to produce quality robustness diagrams, or whether a brief amount of knowledge acquired with a short learning curve would suffice. To this end, this paper presents a student-based experiment to carry out this empirically evaluation.

The remainder of this paper is structured as follows; Section 2 provides a brief description of the robustness analysis technique and the notational constructs of robustness diagrams. In Section 3, the planning and design of the controlled experiment is presented. The experimental results and their analysis are presented in Section 4. Finally, Section 5 concludes and suggests future work.

II. BACKGROUND

In this section we provide the relative background that motivated this research and prompted the need to perform the controlled experiment presented in this paper.

A. *Performing Robustness Analysis and Developing Robustness Diagrams*

As a prelude to describing the relative background that motivated this research, this section presents a brief overview of robustness analysis and diagrams. A more detailed description of the robustness analysis technique and the notational constructs of robustness diagrams are presented in [2, 17, 27,32]. In order to develop a robustness diagram for a use case, the corresponding use case description and domain model are required as input. Domain models are built using the same notational constructs as class diagrams. Domain models however serve a different purpose than traditional class diagrams that are used to implement the system. Domain models are concerned with capturing the business concepts of the problem, while a class diagram is concerned with showing detailed information regarding the final solution. For this reason, a domain model contains far less information than a class diagram and hence it is an artifact which a business analyst is expected to produce, unlike a class diagram. It is very common that domain models evolve into class diagrams, whereby software designers use the domain models produced by the BA (Business Analysts) and "flesh them out" with solution details that will solve the given business problem [11, 20, 28].

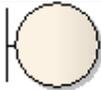
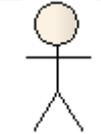
Robustness analysis is performed by applying the following steps:

1. Decompose each narrated use case flow into a set of "steps".
2. Use objects from the domain model and link them together to simulate these steps and realize the use case.

3. Add missing classes and associations in the domain model during step (2).
4. Use terms from the domain model to disambiguate the use case during step (2).
5. Complete other information in use case text that might be missing during step (2).
6. Alternative flows in use case descriptions may optionally be highlighted in the robustness diagram using a different color for distinction from the nominal flow.

The outcome of robustness analysis is a robustness diagram. A single robustness diagram should be developed for each use case [1, 2, 26-28, 32]. Robustness diagrams contain a relatively small set of notational constructs and syntax rules (Table 1).

TABLE 1.
THE NOTATIONAL CONSTRUCTS OF ROBUSTNESS DIAGRAM AND THEIR SYNTAX RULES

Element	Description	Syntax Rules
 Boundary object	Boundary objects represent interfaces that facilitate communication with actors.	Boundary objects can only be associated with actors or control objects.
 Control Object	Control objects are the “muscle and brains” of a system. Control objects interact with boundary and entity objects and perform algorithmic activities to provide services needed by an actor. A control object can coordinate with other control objects to coordinate activities in order to carry out a service. Control objects also check business rules to allow or prevent certain functionalities from executing.	Control objects can only be associated with boundary, entity or other control objects.
 Entity Object	Entity objects store information about real-world concepts. Entity objects should already exist in the domain model.	Entity objects can only be associated with control objects.
 Actor	Any external entity that interacts with the given system.	Actors can only be associated with boundary objects.

Application of the robustness analysis technique is further elaborated using a use case named Register Course. Assume the original description of the Register Course use case and the corresponding domain model is as shown in Figure 1. When analyzing the narrative text of the Register Course use case, it can be deduced that the Student interacts with the system through an interface that displays various courses that could be registered in. The appropriate boundary object to represent this interface would be Course Viewer, which is already

available in the domain model. The Course Viewer object will then forward the request to register in a course along with the required relative information to the Register in Course control object. The required information will include information about the student as well as the course name and type. The two course types are missing from the domain model and therefore the domain model should be updated accordingly (see Fig. 2 middle). The Register in Course object then delegates the responsibility of checking the course prerequisites to the Check Prerequisites control object. If the prerequisites are satisfied, the Register in Course object then invokes the Update Student Record control object to register the Student in the Course and update his Student Record. The statement of updating of the Student Record is missing however from the Register Course use case description and therefore the description should be updated accordingly (see Fig. 2 top). While performing robustness analysis the robustness diagram shown in Figure 2 bottom is constructed.

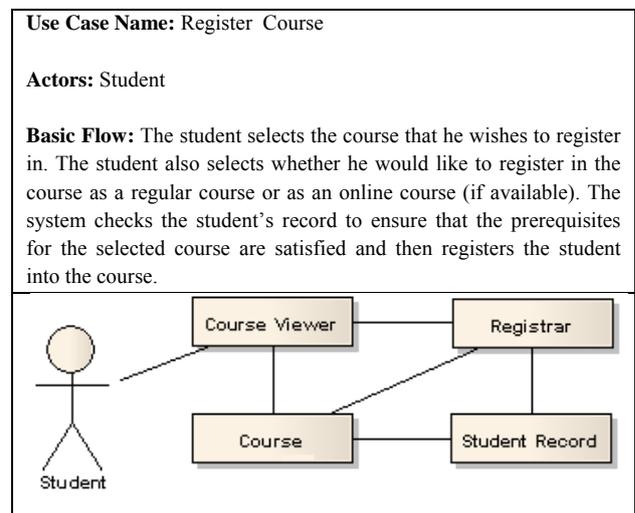
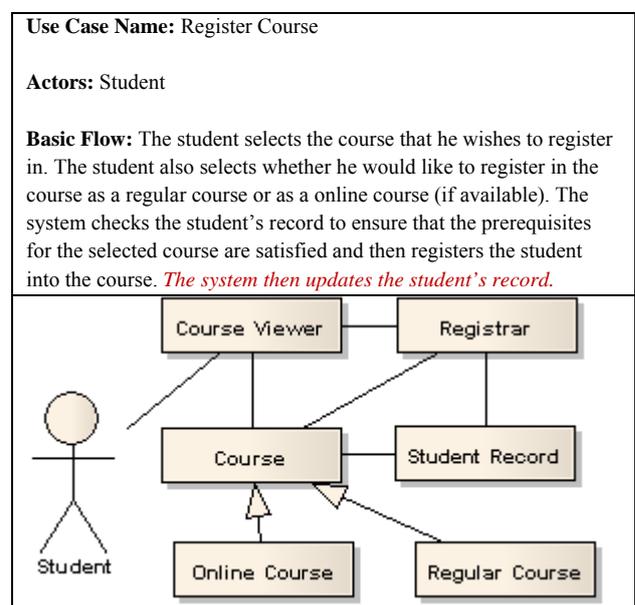


Figure 1 The original Register Course use case description (left) and domain model (right)



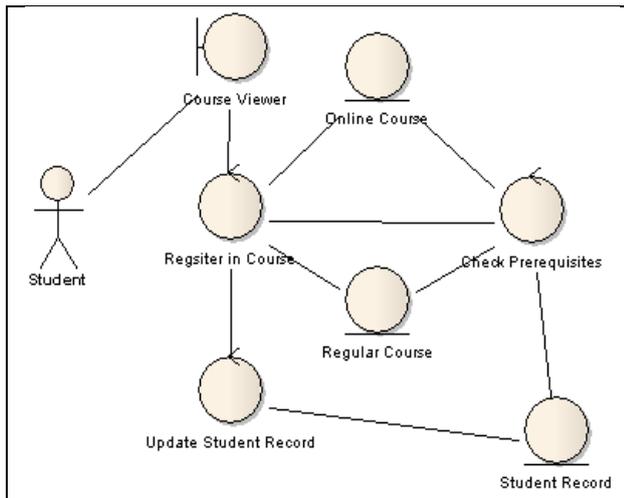


Figure 2 The updated Register Course use case description (top) and domain model (middle) and robustness diagram (bottom)

B. Quality Attributes of Robustness Diagrams

In order to assess the performance of the subjects in developing robustness diagrams, it is important to first identify the quality attributes that should exist in robustness diagrams. It is also important to identify the types of defects that affect each attribute. The literature has proposed many guidelines to perform robustness analysis and produce high quality robustness diagrams [2, 26-28]. For example, [27, 28] outline a set of top ten robustness analysis errors. Such errors include (a) violating the robustness diagram syntax rules, (b) failing to model the use case alternative flows, (c) including too many or too few control objects, (d) including detailed design decisions within the robustness diagram...etc. In [2, 27, 28], a set of heuristics are presented which can be used to determine when an object should be created and its appropriate type. For example, when analyzing the use case text, nouns present candidates for boundary and entity objects, while verbs present candidates for control objects. Based on the literature review, the quality attributes of robustness diagrams were categorized by the author of this paper into three major categories as shown in Table 2.

A robustness diagram lacking any of the quality attributes, presented in Table 2, is likely to lead to defects and harmful consequences. An exhaustive list of defect examples would be very extensive and would need a great deal of space to present. Alternatively, Table 3 outlines a large cross section of defect examples that can affect each quality attribute. Measurement and scoring of these defects are presented in detail in Section 3.8. Many of the defect examples presented in Table 3 were specified explicitly in the literature while others were deduced based on the information provided in the literature.

TABLE 2. QUALITY ATTRIBUTES OF ROBUSTNESS DIAGRAMS

Quality Attribute	Definition
Completeness	The robustness diagram must model all the flows defined in its corresponding use case description. To achieve completeness, a robustness diagram must contain all the necessary entities (actors, objects and associations) that collaboratively realize the entire set of use case flows [2, 26-28].
Fault-Free	A robustness diagram must not contain any information or facts that are incorrect or contradicting to what is stated in its corresponding use case description. Such incorrectness can be in the form of any diagrammatic element which is not stated by a use case description or required to realize a particular use case flow. Incorrectness may also be in the form of a modeled workflow which misrepresents the underlying use case and domain model [2, 26-28].
Design Properness	A robustness diagram must be readable, precise and unambiguous. Readers of the robustness diagram must be able to easily identify the modeled workflows and how they realize the scenarios described in the corresponding use case description. The diagram should also not contain repeated diagrammatic elements as this may lead to confusion. All stakeholders must be able to infer a common understanding of the functional flows presented by the diagrams. The diagram should also not contain detailed design decisions. [2, 26-28].

TABLE 3. ROBUSTNESS DIAGRAMS DEFECT EXAMPLES

Category	Examples
Completeness	<ol style="list-style-type: none"> 1. A missing actor which was stated in a particular flow of a use case description. [26-28] 2. A missing interface object which was required to realize a particular flow of a use case description. [26-28]. 3. A missing control object which was required to realize a particular flow of a use case description. [26-28] 4. A missing entity object which represents data created or accessed as stated in a use case description. [26-28] 5. A missing association between two objects [27, 28] 6. A missing association between an actor and an interface object. [28] 7. A missing collection of actors, objects and their interconnecting associations that would model a flow (or part of a flow) described in a use case. Such flow maybe the use case's basic flow, an alternative, exceptional flow or a sub-flow. [28]
Fault-Free	<ol style="list-style-type: none"> 1. Incorrect Information: [26-28] <ol style="list-style-type: none"> a. An inclusion of an actor that was not described in the corresponding use case description. b. An inclusion of an interface object that represents a type of interface to the

	<p>system which does not represent or satisfy the requirements stated in the corresponding use case.</p> <ul style="list-style-type: none"> c. An inclusion of a control object which symbolizes an action to be performed which does not contribute to representing or satisfying the requirements stated in the corresponding use case. d. An inclusion of a control object which symbolizes an action to be performed by an external entity. e. An inclusion of an entity object that symbolizes a source of information that is not required in order to satisfy the intended goals of the corresponding use case. f. An inclusion of an entity object which symbolizes a source of information available outside the system, unless the diagram shows a means to obtain this information from the external entity. g. An association between an actor and an interface object which should not exist. h. An association between an interface and a control object which should not exist. i. An association between a control and a entity object which should not exist. <p>2. Syntax Errors: [26, 28]</p> <ul style="list-style-type: none"> a. An actor associated with another actor. b. An entity object associated with another entity object. c. An interface object associated with another interface object. d. An interface object associated with an entity object (unless for the purpose of a straightforward information retrieval process). e. An actor associated with a control object. f. An actor associated with an entity object. g. An incorrect graphical representation of an actor using an icon that represents an object. h. An incorrect graphical representation of an interface object using an icon that represents another type of object or an actor. i. An incorrect graphical representation of a control object using an icon that represents another type of object or an actor. j. An incorrect graphical representation of an entity object using an icon that represents another type of object or an actor.
<p>Design Properness</p>	<ul style="list-style-type: none"> 1. Using too many control objects to perform an activity that would ideally be performed by one control object. [2, 28] 2. Using a control object to perform a set of activities that would ideally be performed by a number of control objects. [28] 3. Including detailed design information: [2, 28]

	<ul style="list-style-type: none"> a. Including objects that represent specific solutions. b. Allocating attributes and operations to objects. c. Specifying association end cardinalities. d. Specifying specializing associations, such as aggregation, composition and generalization. e. Specifying association end qualifiers. <p>4. Including repetitive information: [26, 27]</p> <ul style="list-style-type: none"> a. Including repeated objects unless greatly improves the presentation of the diagram. b. Including repeated actors unless greatly improves the presentation of the diagram. c. Failing to reuse objects appropriately to model various flows described in the corresponding use case.
--	--

III. EXPERIMENTAL PLANNING

This section describes a controlled experiment that took place at King Fahd University of Petroleum and Minerals in the Kingdom of Saudi Arabia. The experiment adheres to the experimentation process outlined by Wohlin et al. [33]. In accordance with the process outlined in [33], the subsections below describe the experiment's: definition, context, hypotheses formulation, subject selection, design, instrumentation and measurement techniques, and validity evaluation, respectively.

A. Experiment Definition

The main research question posed by this experiment is whether a strong OO background is required to perform proper robustness analysis and develop quality robustness diagrams in comparison to the robustness diagrams developed with only a brief OO knowledge. Therefore, the only independent variable of this experiment is the in-depth OO design and programming knowledge and hence two groups exist; a group of subjects with strong OO knowledge (OOK) and a group without in-depth OO knowledge (BOOK). This experiment also has three dependent variables upon which the groups are compared. The three dependent variables are based on the three main categories of robustness diagrams quality attributes shown in Table 2; content completeness (C), false facts and information (FF), and design properness (DP).

B. Experiment Context

This experiment involved 2nd and 3rd year Software Engineering undergraduate students. The experiment was conducted during the first semester of the 2009-2010 academic year. It was conducted as in-class exercises and their robustness diagrams were collected at the end of the class. The subjects were made aware that their diagrams will be collected for scoring in order to ensure their committed engagement in the exercises. Upon completion of the experiment, the subjects were notified that robustness analysis will not be covered in any subsequent

exams, assignments or project tasks. The subjects were also later notified that while their diagrams were scored, the scores will not contribute towards their final grades. Before assigning the experimental tasks the subjects were presented with a series of two one-hour lectures that will introduce them to the concepts and techniques of robustness analysis, followed by another series of two one-hour lectures that will allow the subjects to practice these techniques using a number of examples. The experimental tasks were then undertaken in the following two one-hour lectures. The students were not informed about the hypotheses under investigation.

The instructors of the respective courses appreciated the value of robustness analysis and developing robustness diagrams at the start of the semester. The learning value that the subjects were intended to receive by the courses hence included robustness analysis regardless of the experiment being conducted. In fact, robustness analysis was included in subsequent offerings of the courses.

C. Hypotheses Formulation

Three hypotheses were produced to account for performance of the groups in developing robustness diagrams (see Table 4). The alternative hypotheses (H_a) for the Completeness (C) variable states that the robustness diagrams developed by OOK subjects will model more factual information in comparison with the robustness diagrams developed by BOOK subjects. The alternative hypotheses (H_a) for the Fault-Free (FF) and Design Properness (DP) variables state that OOK subjects will commit less Fault-Free and Design Properness errors than BOOK subjects. Therefore, all variables are one-tailed hypotheses.

TABLE 4.
THREE DEPENDENT VARIABLES AND THEIR
CORRESPONDING HYPOTHESES

Dependent Variable	Null Hypothesis (H_o):	Alternative Hypothesis (H_a):
Completeness	$(H_o1): C(OOK) \leq C(BOOK)$	$(H_a1): C(OOK) > C(BOOK)$
Design Properness Violations	$(H_o2): DP(OOK) \geq DP(BOOK)$	$(H_a2): DP(OOK) < DP(BOOK)$
Fault-Free Violations	$(H_o3): FF(OOK) \geq FF(BOOK)$	$(H_a3): FF(OOK) < FF(BOOK)$

D. Subject Selection

In total, 54 students participated in the experiment. Informal interviews with the subjects have indicated that none of them had previous exposure to robustness analysis. It is beneficial that the subjects did not have prior experience with robustness analysis modeling as there might be a tendency by the subjects to ignore the prescribed techniques and concepts taught in the lectures, and instead apply the techniques that are more familiar to them from their experience. However, it must be noted that the fact that the subjects lack any robustness analysis and modeling experience may raise concern with respect

to external validity. This issue is discussed in more detail in Section 4.2.4. One group consists of 28 students who were enrolled in a second year course called “Introduction to Software Engineering”. The other grouped consists of 26 students who were enrolled in a third year course called “Software Design and Architecture”. Both sets of subjects are enrolled in the undergraduate Software Engineering program and therefore their relative educational experience was known beforehand. Both sets of subjects were distinct, meaning no subject was enrolled in both courses at the same time. The experiment was conducted at an early stage of the first semester of the academic year. Therefore, the relative educational experience of the second year group of subjects is determined to consist of one course that introduces them to the basic concepts of programming, such as primitive data types, loops, conditions, functions, user input handling, and basic read/write operations using text files. The relative educational experience of the third year group of subjects subsumes that of the second year group in addition to two advanced programming courses which cover advanced OO concepts and data structures. The third year group of subjects has also undertaken a course which is mainly concerned with OO modeling using UML.

E. Experimental Design and Tasks

The subjects were divided into two groups (BOOK and OOK) based on the course they are enrolled in. Given their educational background, subjects enrolled in the second year course would resemble a population that lacks in-depth OO knowledge (the BOOK group), while subjects enrolled in the third year course would resemble technical personnel with strong OO knowledge (the OOK group).

In this experiment, the subjects were required to consider two distinct use cases along with their corresponding domain models. The use cases are named “Filter Restaurants” [26] and “Place and Order” [2]. The ideal robustness diagrams for both these use cases are presented in their respective sources. Using use cases from external sources is important to eliminate biases even though the author of this paper is not the creator of the robustness analysis technique. For each part of this experiment, the subjects were given the use case descriptions and the use case diagrams of the systems from which the use cases have originated. The subjects were also provided the domain models for each system. The subjects were then asked to develop a robustness diagram for each of the two use cases. All artifacts used in the experiment such as use case descriptions, the use case diagram and their corresponding domain models can be found in Appendix A. Prior to beginning each exercise, the subjects were introduced to the business related to each use case so that they would understand the context in which each use case is executing.

To mitigate the effect of individual and group abilities as well as system learning effects, the experimental design shown in Table 5 was used. The order of the use cases given was similar to both groups. Use case “Filter Restaurants” was chosen at random to be given to the

subjects first. Learning effects based on the order of use cases is mitigated since students work with the same use case first (“Filter Restaurants”) before working with the “Place an Order” use case. Meaning that both groups had the same chance to learn about developing robustness diagrams since after “Part 3” of the experiment both groups would have been exposed to the same material and exercises. An important factor to take into consideration is that the OOK group, given their prior knowledge of OO concepts, is expected to be quicker in learning how to develop robustness diagrams since robustness diagrams is chiefly a simpler version of UML collaboration diagrams which they are already very familiar with through their coursework. Although students of the OOK group have never been exposed to robustness diagrams, they have practically not learned anything new by taking part of this experiment other than learning not to focus on detailed syntax. Therefore this factor in the experiment is acceptable since the reason behind their quicker learning is their in-depth OO knowledge, which is the sole independent variable to the experiment.

Table 6 shows the structural and content details of both use cases as presented in their respective sources. Both use cases contain a total of 19 elements (actors and objects). The “Place an Order” use case however was determined to be a larger use case as it contains 5 more functional facts than the “Filter Restaurants” use case.

TABLE 5.
EXPERIMENTAL DESIGN

	BOOK Group	OOK Group
Part 1	Introduction to robustness analysis and robustness diagrams 2 lectures (approx. 2 hours total)	
Part 2	Robustness analysis practice using various examples	
Part 3	Develop Robustness diagram for: “Filter Restaurants” use case	Develop Robustness diagram for: “Filter Restaurants” use case
Part 4	Develop Robustness diagram for: “Place an Order” use case	Develop Robustness diagram for: “Place an Order” use case

TABLE 6.
DETAILS OF THE TWO USE CASES USED IN THIS EXPERIMENT

	Filter Restaurants	Place an Order
# of actors	1	1
# of interface objects	4	6
# of control objects	9	7
# of entity objects	5	5
# of functional facts	12	17

Time Allocation

As the exercises were relatively small, subjects were expected to finish them in approximately 30 minutes (±15 minutes). Subjects did not have to face any timing pressures since lectures are approximately 1 hour long.

All subjects finished their tasks within 30 minutes and no great time differences were observed.

F. Instrumentation

The subjects were required to create their robustness diagrams using pencil and paper only to eliminate any effect that might be introduced by tool support.

G. Analysis Procedure

The quantitative data presented in this paper is considered as discrete count as it is assumed that all deficiencies have an equal unit weighting. The various data sets were examined for their compliance to normality assumptions using the Shapiro-Wilk test [29]. This test was chosen instead of other common normality tests, such as Kolmogorov-Smirnov [16] and Anderson-Darling [3], since it does not require that mean or variance of the hypothesized normal distribution to be specified in advance. Therefore, the Shapiro-Wilk test can be considered to be more powerful than other common normality tests. Further details of the Shapiro-Wilk test can be found in [29]. Executing the Shapiro-Wilk test revealed that a significant subset of the datasets are non-normal. Therefore, to adopt a conservative approach, the quantitative analysis performed will consider all datasets as being sampled from non-parametric distributions.

H. Scoring and Measurement

Table 7 presents the scoring strategy of defects affecting each quality attribute. Scoring of the robustness diagrams was conducted independently by three authors of this paper. Each defect was recorded by each author and discrepancies were resolved through verbal discussion.

TABLE 7.
SCORING STRATEGY OF ROBUSTNESS DIAGRAM DEFECTS

Category	Scoring Strategy
Completeness	All “Completeness” defects are scored similarly (as a discrete count ‘1’) regardless of their type. If a diagram does not model a given fact-A then this is scored as the sum of missing actors, objects and associations that represent that fact. The missing entities are determined based on the ideal solutions provided in [2, 25-28].
Fault-Free	“Fault-Free” is scored as the sum of unique “Fault-Free” defects committed in the use case model. All “Fault-Free” defects are scored (as a discrete count ‘1’) regardless of their type. For example, if the robustness diagram shows an actor associated with two different control objects, then these counts as two “Fault-Free” defects. For example, if the diagram shows that information is retrieved from a Savings Account entity object when the use case states that information is retrieved from a chequeing account, then this counts as one “Fault-Free” defect as well as one “Completeness” defect. [25-28]
Design Properness	All “Design Properness” defects are scored similarly (as a discrete count ‘1’) regardless of their type. In the case of having too many or too

	<p>few control objects that should be combined into one control object, defects are scored based on the number of fine-grained or coarse-grained control objects, respectively. In the case of unnecessarily repeated information, for example, a repeated entity object or actor, defects are based on the number of unnecessary repetitions. Instances of having highly detailed design decisions in the diagram are scored based on the object containing the extraneous details. [2 , 25-28]</p>
--	--

IV. ANALYSIS AND INTERPRETATION

A descriptive summary for each non-parametric variable is presented in terms of a notched box and whiskers plot (see Fig. 3). The upper and lower horizontal lines show the upper and lower quartiles respectively. The median is indicated by the middle horizontal line. The confidence interval around the median is shown using tilted lines stemming from the median. Whiskers (vertical lines) extending from the notched boxes extend to the furthest observations within ±1.5 IQR (interquartile ranges) of the 1st and 3rd quartile. Observations outside 1.5 IQRs are marked as near outliers (+), and those outside 3.0 IQRs are marked as far outliers (*).

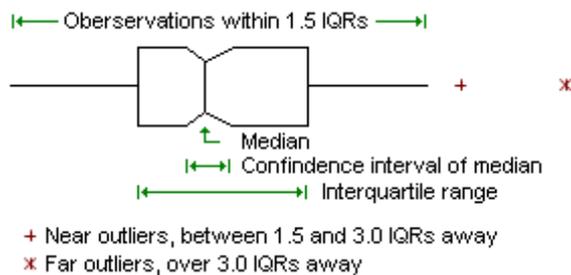


Figure 3 Illustration of the box and whiskers plot's diagrammatic notation

The Mann-Whitney U statistic (of the 1st sample) was used to test differences between the medians of related samples. More information about the Mann-Whitney test is available in [30]. The presence of a number of ties within the datasets prevents us from using an exact test and hence the probability provided should in general be considered as an underestimation. All confidence intervals around the difference between medians are given at the standard 95% level and were computed using the Hodges-Lehman method [20]. Furthermore, we will provide an estimate of the size of the difference between the two groups by estimating the associated effect size. In this paper, we use Cliff's delta [6-8] as a non-parametric effect size measure. It was empirically demonstrated by Hess et al. [12] and Kromrey et al. [18, 19] that when the data is non-normal or possesses variance heterogeneity,

Cliff's delta is superior to Hedges' g and Cohen's d. Cliff's delta examines the probability that individual observations within one group are likely to be greater than the observations in the other group:

$$\Delta = \Pr(x_{i1} > x_{j2}) - \Pr(x_{i1} < x_{j2})$$

Where x_{i1} is a member of population one and x_{j2} is a member of population two.

While there are three methods of inference of Cliff's [8], we will utilize the "consistent" estimate of the variance as it allows the construction of the associated asymmetric confidence intervals around at 95%. The choice of variance procedure has been empirically proven by Kromrey and Hogarty [18] to be relatively unimportant across a wide range of circumstances. In this article, we will utilize the effect size measure to compute exploratory significance hypothesis testing to further confirm the results obtained from the Mann-Whitney test. For the two groups involved in our controlled experiment, if the confidence interval only includes positive numbers then OOK > BOOK (favoring OOK subjects); if it only includes negative numbers then BOOK > OOK (favoring BOOK subjects); if the confidence interval includes zero, the populations are considered equal.

A. Performed Analysis

The analysis performed investigates the performance of the two groups with respect to each quality attribute in isolation using both use cases: in Section 4.1.1, the performance of the groups with respect to the "Completeness" quality attribute; in Section 4.1.2, the performance of the groups with respect to "Fault-Free" quality attribute, and in Section 4.1.3, the performance of the groups with respect to the "Design-Properness" quality attribute.

BOOK vs. OOK – Completeness

Figures 4 and 5 show the results of the cumulative "Completeness" count from the "Filter Restaurants" and "Place an Order" use cases, respectively. Table 8 shows that no statistically significant difference was observed between the performances of the two groups with either use case. This indicates that both groups modeled relatively the same amount of information stated in the use case descriptions within their robustness diagrams. This statistical insignificance is further confirmed as the confidence interval around includes the value zero for both use cases (see Table 9).

TABLE 8.
MANN-WHITNEY TEST FOR THE ‘COMPLETENESS’ RESULTS

Alternative Hypothesis - (H_a): C (OOK) > C (BOOK)								
Use Case	Subjects	Rank sum	Mean rank	U	Median difference	95.3% CI	Mann-Whitney U statistic	1-tailed p
Filter Hotels	OOK	784.0	30.15	295.0	1.0	0.0 to $+\infty$	295.0	0.1117
	BOOK	701.0	25.04	433.0				
Place an Order	OOK	704.0	27.08	375.0	0.0	-1.0 to $+\infty$	375.0	0.5767
	BOOK	781.0	27.89	353.0				

TABLE 9.
CLIFF’S DELTA FOR THE ‘COMPLETENESS’ RESULTS

System	Cliff’s delta ($\hat{\delta}$)	Variance	Confidence Interval around delta ($\hat{\delta}$)	
			maximum	minimum
Filter Hotels	0.192	0.027	0.481	-0.134
Place an Order	-0.154	0.043	0.245	-0.508

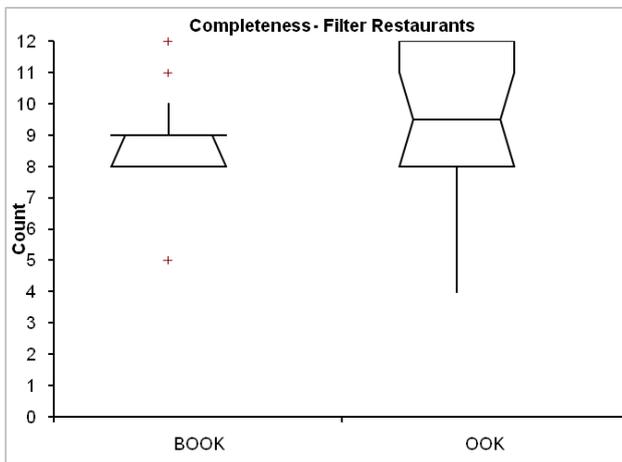


Figure 4 BOOK vs. OOK – Completeness (Filter Restaurants)

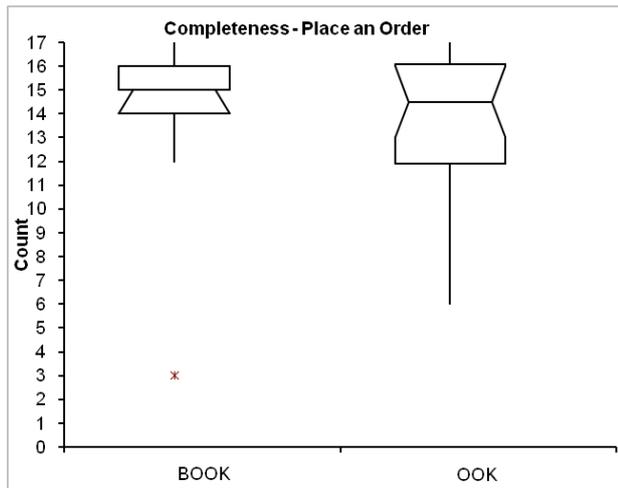


Figure 5 BOOK vs. OOK – Completeness (Place an Order)

did not hinder them from introducing the appropriate objects needed to realize the use case scenarios. Moreover, the BOOK subjects were able to correctly consider and introduce the appropriate relationships between the objects they used in their diagrams which would accurately reflect the use case scenarios.

BOOK vs. OOK – Fault-Free

Figures 6 and 7 show the results for the ‘Fault-Free’ quality attribute with respect to the “Filter Restaurants” and “Place an Order” use cases, respectively. The results do not show a statistically significant difference between the performance of the BOOK and OOK subjects (Tables 10 and 11 with respect to the “Filter Restaurants” use case. However, the results show a statistically significant difference between the performances of the groups with respect to the “Place an Order” use case. The positive range of the confidence interval around (Table 11) indicates that OOK subjects have performed better than the BOOK subjects.



Figure 6 BOOK vs. OOK – Fault-Free (Filter Restaurants)

Discussion – According to the results, it can be inferred that in-depth knowledge of OO concepts does not have a statistically significant effect on the ability of the subjects to account for possible use case scenarios in their design. The lack on in-depth OO knowledge of BOOK subjects

TABLE 10.
MANN-WHITNEY TEST FOR THE 'FAULT-FREE' RESULTS

Alternative Hypothesis - (H _{a2}): FF (OOK) < FF (BOOK)								
Use Case	Subjects	Rank sum	Mean rank	U	Median difference	95.2% CI	Mann-Whitney U statistic	1-tailed p
Filter Hotels	OOK	695.0	26.73	384.0	0.0	-∞ to 0.0	384.0	0.3597
	BOOK	790.0	28.21	344.0				
Place an Order	OOK	548.0	21.08	531.0	-1.0	-∞ to 0.0	531.0	0.0013
	BOOK	937.0	33.46	197.0				

TABLE 11.
CLIFF'S DELTA FOR THE 'FAULT-FREE' RESULTS

System	Cliff's delta ($\hat{\delta}$)	Variance	Confidence Interval around delta ($\hat{\delta}$)	
			maximum	minimum
Filter Hotels	0.055	0.106	0.317	-0.215
Place an Order	0.478	0.023	0.716	0.141

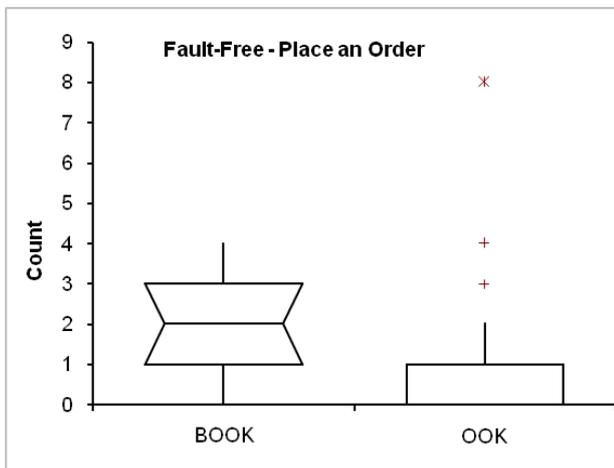


Figure 7 BOOK vs. OOK – Fault-Free (Place an Order)

use case (see Table 6), which in turn would be modeled using a larger robustness diagram. Naturally, it is more likely for syntax errors to be introduced as the robustness diagram becomes larger. Therefore, it is expected that as the size of robustness diagrams increase, the BOOK subjects will increasingly introduce more syntax defects than their OOK counterparts.

BOOK vs. OOK – Design Properness

Figures 8 and 9 show the results of the cumulative “Design Properness” violations count from the “Filter Restaurants” and “Place an Order” use cases, respectively. The difference between the performances of the two groups with either use case was statistically insignificant (Table 12). Table 13 further corroborates that the BOOK group would not commit a statistically significant larger number of improper design decisions than the OOK group.

Discussion – Further examination of the subjects’ performances was conducted to shed more “light” into this situation. The ‘Fault-Free’ category consists of two subcategories: ‘Incorrect Information’ and ‘Syntax Errors’. It was revealed that the BOOK subjects performed poorly in the “Syntax Errors” category as they committed an average of 1.36 errors in comparison to an average of only 0.38 errors by the OOK subjects. This might be attributable to the fact the OOK subjects have taken a course that is mainly concerned with modeling using UML in which they were trained to develop syntactically more complicated diagrams (such as sequence, collaboration and activity diagrams) and were taught to carefully abide to their syntax rules. On the other hand, the BOOK subjects had not at the time of the experiment undertaken such course. In fact, at the time of the experiment, the BOOK subjects had no prior training in developing any software models. Hence, the BOOK subjects will naturally be less mindful of syntax rules in comparison to their OOK counterparts. In this experiment, a statistical significance between the performances of the two groups was observed only with the “Place an Order” use case. This is most likely because the “Place an Order” use case is a larger use case than the “Filter Restaurants”

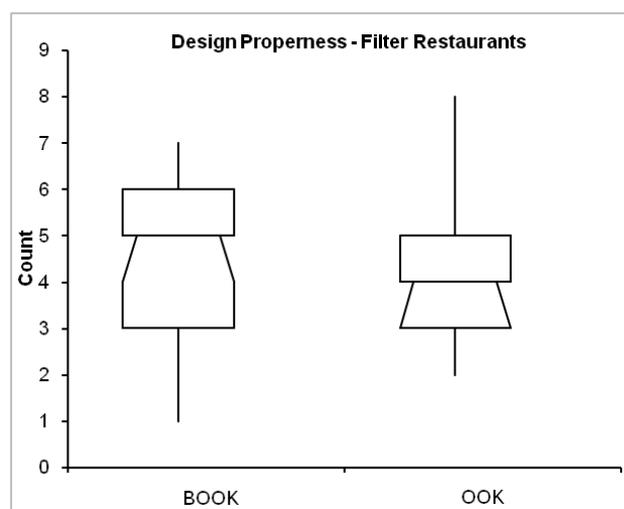


Figure 8 BOOK vs. OOK – Design Properness (Filter Restaurants)

TABLE 12.
MANN-WHITNEY TEST FOR THE 'DESIGN PROPERNESS' RESULTS

Alternative Hypothesis - (H _{a3}): DP (OOK) < DP (BOOK)								
Use Case	Subjects	Rank sum	Mean rank	U	Medians difference	95.3% CI	Mann-Whitney U statistic	1-tailed p
Filter Hotels	OOK	659.0	25.35	420.0	-1.0	-∞ to 0.0	420.0	0.1624
	BOOK	826.0	29.50	308.0				
Place an Order	OOK	697.0	26.81	382.0	0.0	-∞ to 1.0	382.0	0.3766
	BOOK	788.0	28.14	346.0				

TABLE 13.
CLIFF'S DELTA FOR THE 'DESIGN PROPERNESS' RESULTS

System	Cliff's delta ($\hat{\delta}$)	Variance	Confidence Interval around delta ($\hat{\delta}$)	
			maximum	minimum
Filter Hotels	0.154	0.034	0.476	-0.205
Place an Order	0.077	0.037	0.420	-0.286

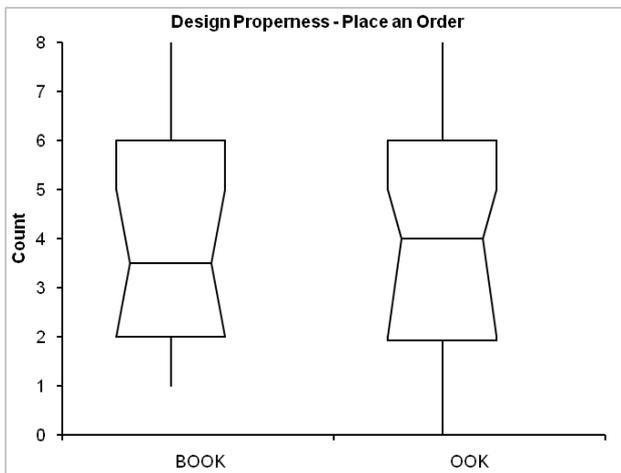


Figure 9 BOOK vs. OOK – Design Properness (Place an Order)

Discussion – According to the results, it can be inferred that in-depth knowledge of OO concepts does not have a statistically significant effect on the ability of the subjects to adhere to the design guidelines of robustness diagrams. The lack on in-depth OO knowledge of BOOK subjects did not hinder them from determining the correct level of granularity of control objects (i.e. avoid creating control objects that do too much or too little). The readability, clarity and preciseness of robustness diagrams created by BOOK subjects are at similar quality levels as the robustness diagrams developed by their OOK counterparts. BOOK subjects were able to avoid introducing repeated diagrammatic elements. The robustness diagrams developed by BOOK subjects clearly showed the workflows of the use case descriptions and how they were realized.

B. Threats to Validity

In this section we present threats to the validity of the study in accordance with the standard classification [33].

Conclusion Validity

In any student-based experiment, individual abilities influence the observed. If there is a large degree of heterogeneity within the subjects, the variations in the observed results can be due to individual differences rather than the prescribed techniques. To mitigate against this serious validity threat, it is necessary to increase homogeneity within the subjects. To this end, the experiment was conducted with the entire list of students registered in both courses. As a result, both groups embodied a similar spectrum of individual abilities. This was confirmed by reviewing the subjects' academic standings before the experiment. All subjects were undergraduate Software Engineering students and thus their educational interests are assumed to be the same. All subjects were never exposed to robustness analysis or robustness diagrams before this experiment; and whom all underwent the same classes and practice. An advantage of choosing subjects who were not exposed to robustness analysis and robustness diagrams prior to the experiment is that it ensures that the subjects applied the prescribed methods instead of techniques they might have learned previously. A similar argument was also set forth by the authors of [4] about the advantage of using students as subjects in controlled experiments instead of professionals.

Internal Validity

To combat any fatigue or maturation threats, subjects were allotted one hour in order to complete each experimental task that would usually last approximately 30minutes (±15 minutes). When the experiment was conducted, all students completed their experimental tasks in no longer than 30 minutes. Therefore, it can be inferred that the subjects did not feel any significant time pressure to complete the tasks.

To mitigate against self-selection, population selection was based on subjects registered in two different courses. The academic standings and ability levels of the subjects were evenly distributed. That is, each course contained subjects with high, average and below-average academic standings. It can be argued that the skill levels of business

analysts and software engineers also vary. Therefore, the skill levels of the subjects reflect the range of skill levels of business analysts and software engineers. To mitigate against morality threats, the subjects were under the influence that robustness analysis is a technique which they will be evaluated on during the course (by means of a homework assignment, quiz, exam...etc). Naturally, it is expected that the subjects will be motivated to learn from and participate in the experiment in order to perform well when their robustness analysis skills are later evaluated, as they would with all other course materials. The subjects were only notified that they will not be evaluated on robustness analysis after the experiment was completed. All subjects were undergraduate Software Engineering students and thus the context of the experiment falls under their natural learning interests.

Construct Validity

The design of this experiment aimed to minimize the construct validity of the dependent variables. To minimize the effects of individual capabilities, the entire list of students registered in two different courses were selected to form two groups; system differences and ordering effects were minimized by having both groups perform their initial robustness analysis exercise on the same use case before subsequently performing their second robustness analysis on the other use case. In this experiment, the "Filter Restaurants" use case was randomly chosen to be the first use case for the subjects to apply robustness analysis. Biasness towards any group with respect to the use cases used in the experiment was eliminated by using two use cases provided by two different authors, who have no connection with this experiment.

External Validity

As with any experiment conducted using students as subjects, it is unsafe to generalize these result of the experiment to software professionals. In the case of our experiment, it is unsafe to generalize the performance of the second and third students to business analysts and software engineers, respectively. Therefore, the populations were chosen not to represent business analysts and software engineers. The populations were chosen to represent a group with in-depth OO knowledge and one without in-depth OO knowledge. The scope of the experiment is not to determine the performance of business analysts vs. software engineering professionals. The scope of the experiment was focused to determine whether in-depth knowledge of OO concepts is required to develop quality robustness diagrams.

Another inherent external validity is that the experiment was conducted with relatively small artifacts, although both use cases utilized in this experiments was applied in an industrial setting [2, 26]. However, it is generally unsafe to generalize the results of our experiment to full-scale industrial settings and artifacts. This experiment considered only one use case from their

respective systems. There may be an effect when considering the entire set of use cases of a given system and considering a much larger domain model when developing robustness diagrams as many objects may be common and reused by several use cases. This potential effect was not investigated by this experiment. Industrial use cases are generally larger and represent more complex functionalities, thus they are more likely to contain more factual information and alternative flows that needs to be modeled in their respective robustness diagrams. Developing the robustness diagram of an industrial use case would therefore require a larger quantity of objects to realize as well as associations between those objects and hence there is a greater vulnerability to committing mistakes.

VI. CONCLUSION

Robustness analysis and robustness diagrams can be a very valuable tool in the software development life cycle. Robustness diagrams can be developed at an early stage of the software development life cycle and can be used to obtain early customer feedback. Robustness diagrams can also be used as basis to develop a more comprehensive set of acceptance, system, integration and unit tests. Robustness analysis relieves its users from being overly concerned with too many syntax rules and in turn they will be able to consider a wider spectrum of design alternatives. Robustness analysis minimizes the gap between the analysis and design phases increasing the potential of developing a system that satisfies its requirements. Robustness analysis can be used to improve the completeness of use case descriptions and domain models and enhance their consistency. However, these benefits may only be reaped if robustness analysis is performed at a very early stage in the development life cycle and hence it is required to be performed by business analysts whom are assumed not to have OO knowledge which is a core requisite to developing robustness diagrams. Business analysts may overcome this limitation by acquiring brief knowledge of OO concepts (through a short course). Naturally, the success gained from using the developed robustness diagrams depends on their quality. Therefore, it is not only important that business analysts produce robustness diagrams, but it also crucial that they develop high quality robustness diagrams.

In this paper a subject-based controlled experiment is presented which explores an important research question. The research question posed by this experiment is to evaluate the ability of business analysts, whom acquire a brief level of OO knowledge, to perform robustness analysis and develop robustness diagrams. In order to answer this research question, the quality of robustness diagrams developed by a group that lacks in-depth OO knowledge needs to be compared to the quality of robustness diagrams developed by a group that possess in-depth OO knowledge and whom are technically equipped to develop high quality robustness diagrams.

This experiment was conducted during the lecture time of two distinct undergraduate Software Engineering courses and involved two distinct groups of

undergraduate students as subjects. One group represents a population that lacks in-depth OO knowledge while the other group represents a population that has in-depth OO knowledge. Subjects from both groups were provided two sets of use case descriptions and their corresponding domain models and were asked to perform robustness analysis in order to produce robustness diagrams. The results of this experiment showed no statistically significant difference between the performances of the groups with respect to the completeness and design properness levels of the developed robustness diagrams. No statistically significant improvement was observed between the performances of the groups with respect to the correctness level in the robustness diagrams in one of the two tasks. The experiment results suggest that as the size of robustness diagrams increase, the BOOK subjects will increasingly introduce more syntax defects than their OOK counterparts. However, this issue should not prevent potential users of robustness diagrams who lack in-depth OO knowledge, such as business analysts. This issue can be easily remedied using automated tools that can detect syntax errors in robustness diagrams developed by business analysts.

All subjects finished their tasks under 30 minutes meaning that the subjects had at least 30 more minutes to spare. The difference between the times that the BOOK and OOK subjects required to finish their exercises was not significant. Through informal post-interviews, the subjects have indicated that robustness analysis and robustness diagrams were “not hard to learn and apply”. The students have also indicated that they felt that robustness analysis was the first modeling technique which they have learned that prompted them to consider alternative design solutions rather than looking for the “one and only correct solution”. After the experiment most students were keen to know about the ideal solutions provided in [26] and [2] so that they would be able to compare them with their solutions. Students have also indicated that they feel that robustness analysis should become a permanent component in at least one undergraduate Software Engineering course. Most comments received during the sessions were requests to clarify minor details in the use case descriptions. Overall, the subjects were able to apply the prescribed techniques to produce high quality robustness diagrams without any obvious problems.

APPENDIX A EXPERIMENT ARTIFACTS

A.1 “Filter Restaurants” use case

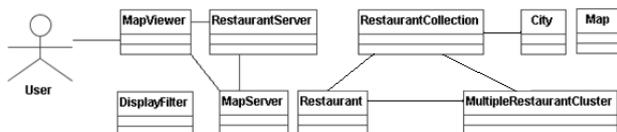


Figure 10 Domain model the RestoMapper system presented in [26]

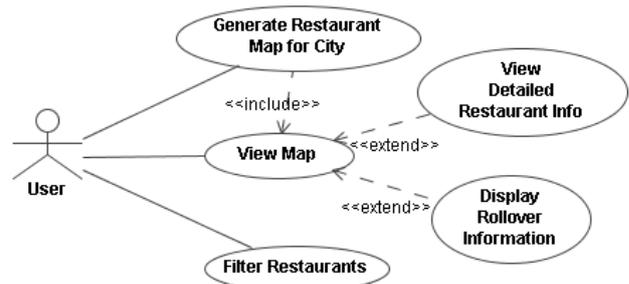


Figure 11 Use case diagram for the RestoMapper system presented in [26] that include the “Filter Restaurants”

Use Case: Filter Restaurants

Basic Flow - Filter by Features:

A list of features is displayed for the User. The User can select one or more features such as the availability of valet parking, live music and a smoking section. A RestaurantFilter is then created by the MapViewer based on the selected features. The MapViewer queries the RestaurantCollection already created and filters it according to the RestaurantFilter. The map is then refreshed to display the filtered restaurants.

Alternative Flow - Filter by Restaurant Chain:

A list of restaurant chains is displayed for the User. The User then selects a particular restaurant chain from the list. A RestaurantFilter is then created by the MapViewer based on the selected features. The MapViewer queries the RestaurantCollection already created and filters it according to the RestaurantFilter. The map is then refreshed to display the filtered restaurants.

Alternative Flow: No hotels matching criterion

If no restaurants matched the filter criterion, the following popup message is displayed to the User: “No restaurants meet filter criterion. Please expand your search”.

Figure 12 The textual description of the “Filter Restaurants” use case.

A.2 “Place an Order” Use Case

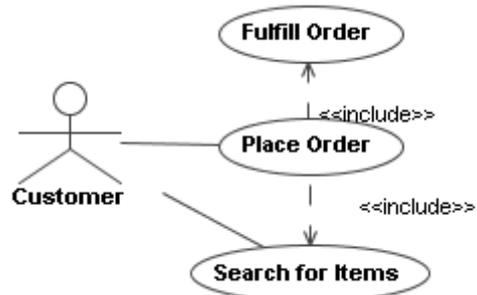


Figure 13 Use case diagram of the system presented in [2] that includes the “Place an Order” use case.

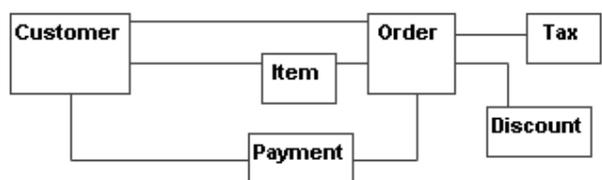


Figure 14 Domain model of the system presented in [2]

Use Case: Place an Order	
Basic Flow	
1.	The customer searches for items via the use case Search for Items.
2.	The use case begins when a customer chooses to place an order from the "Search for Results" page.
3.	The customer adds an order item to their order.
4.	The customer indicates the number of a given item they wish to order.
5.	The system calculates the subtotal for the item by multiplying the unit price by the number ordered.
6.	The customer repeats steps 2 through 5 as necessary to build their order.
7.	The customer provides their shipping and billing information, including their name, phone number, and address.
8.	The system calculates the subtotal for the entire order by adding the subtotals of the individual line items.
9.	The system calculates the taxes applicable for the order.
10.	The system calculates applicable discounts for the order.
11.	The system displays the applicable taxes and discounts.
12.	The system calculates the grand total for the order by adding the applicable taxes to the order subtotal and subtracting the discounts.
13.	The system displays a summary of the order.
14.	The customer verifies that the order is what they want.
15.	The system schedules the order for fulfillment (done by use case Fullfill Order)
16.	The system produces a receipt for the customer that summarizes the order and sends to the customer via email.

Figure 15 The textual description of the "Place an Order" use case

ACKNOWLEDGMENT

The authors would like to acknowledge the support provided by the Deanship of Scientific Research (DSR) at King Fahd University of Petroleum and Minerals (KFUPM) for funding this work through project No. IN111028.

REFERENCES

- [1] K. Aguanno: *Managing Agile Projects*. Multi-Media Publications Inc. 2005.
- [2] S. Ambler: *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*. Wiley. 2002.
- [3] T. W. Anderson ; D. A. Darling: Asymptotic theory of certain "goodness-of-fit" criteria based on stochastic processes. *Annals of Mathematical Statistics* 23: 193–212. 1952.
- [4] E. Arisholm, L. Briand, S. Hove, and Y. Labiche, "The Impact of UML Documentation on Software Maintenance: An Experimental Evaluation," *IEEE Transaction on Software Engineering*, vol. 32, pp. 365-381, 2006.
- [5] Change Vision, Inc. *astah* UML, Ver 6.2.1*. Available at <http://astah.change-vision.com/en/product/astah-uml.html> [Online]. Last Accessed October 2010.
- [6] N. Cliff, *Ordinal Methods for Behavioral Data Analysis*. Lawrence Erlbaum Associates, 1996.
- [7] N. Cliff, "Dominance statistics: Ordinal analyses To Answer Ordinal Questions," *Psychological Bulletin*, vol. 114, pp. 494-509, 1993.
- [8] N. Cliff, "Answering Ordinal Questions With Ordinal Data Using Ordinal Statistics," *Multivariate Behavioral Research*, vol. 31, pp. 331-350, 1996.
- [9] P. Dugerdil and S. Jossi: *Reverse-Architecting Legacy Software Based on Roles: An Industrial Experiment*. *Communications in Computer and Information Science*, Volume 22, Part 2, II, 114-127, 2009.
- [10] Sparx Systems. *Enterprise Architect, Version 8*. Available at <http://www.sparxsystems.com.au/> [Online]. Last Accessed October 2010.
- [11] H. Eriksson, M. Penker, B. Lyons, D. Fado : *UML 2 Toolkit*. Wiley Publishing. 2004.
- [12] M. R. Hess, J. D. Kromrey, J. M. Ferron, K. Y. Hogarty, and C. V. Hines, "Robust Inference in Meta-Analysis: An Empirical Comparison of Point and Interval Estimates Using the Standardized Mean Difference and Cliff's Delta," Annual meeting of the American Educational Research Association, pp. 36, available at: www.coedu.usf.edu/main/departments/me/documents/RobustMeta-AnalysisAERA2005.pdf, 2005.
- [13] M. Host, B. Regnell, and C. Wohlin, "Using Students as Subjects – A Comparative Study of Students and Professionals in Lead-Time Impact Assessment," *Empirical Software Engineering*, vol. 5, pp. 210-214, Nov. 2000.
- [14] International Institute of Business Analysis, "BABOK: Business Analyst Body of Knowledge". [Online www.iiba.org]. Last Accessed October 2011.
- [15] I. Jacobson, *Object-Oriented Software Engineering. A Use Case Driven Approach*. Addison-Wesley, 1992.
- [16] A. Kolmogorov: "Sulla determinazione empirica di una legge di distribuzione". *G. Inst. Ital. Attuari*, 4, 83. 1933.
- [17] A. Kossiakoff and W. N. Sweet: *Systems Engineering Principles and Practice*. John Wiley & Sons, Inc. 2003.
- [18] J. Kromrey and K. Hogarty, "Analysis Options For Testing Group Differences On Ordered Categorical Variables: An Empirical Investigation Of Type I Error Control And Statistical Power," *Multiple Linear Regression Viewpoints*, vol. 25, pp. 70–82, 1998.
- [19] J. Kromrey, K. Hogarty, J. Ferron, C. Hines, and M. Hess, "Robustness in Meta-analysis: An Empirical Comparison of Point and Interval Estimates of Standardized Mean Differences and Cliff's Delta," *American Statistical Association 2005 Joint Statistical Meetings*, pp.7; available at: luna.cas.usf.edu/~mbrannic/files/meta/Robust%20Estimate%20s.pdf, 2005.
- [20] P. Laplante: *Real-time Systems Design and Analysis: 3rd Edition*, Wiley-IEEE Press. 2004.
- [21] E. L. Lehmann, *Non-Parametrics: Statistical Methods Based On Ranks*, Revised. Pearson, 1998.
- [22] No Magic, Inc. . *MagicDraw UML, Ver 16.9*. Available at <http://www.magicdraw.com/newandnoteworthy/magicdraw/16.9> [Online]. Last Accessed October 2010.
- [23] ModelMaker Tools BV. *ModelMaker, Ver 11*. Available at <http://www.modelmakertools.com/modelmaker/index.html> [Online]. Last Accessed October 2010.
- [24] S. Mukhtar: *Applying Robustness Analysis on the Model–View–Controller (MVC) Architecture in ASP.NET Framework, using UML*. Available at <http://www.codeproject.com/KB/architecture/ModelViewC ontroller.aspx> [Online]. Last Accessed Oct 2010.
- [25] D. Rosenberg and K. Scott: *Use Case Driven Object Modeling with UML*. Addison-Wesley, 1999.
- [26] D. Rosenberg, M. Stephens, and M. Collins-Cope :*Agile development with ICONIX process: people process and pragmatism*. Apress, Berkely. 2005.
- [27] D. Rosenberg and M. Stephens, *Use Case Driven Object Modeling with UML: Theory and Practice*, Apress, 2007.

- [28] K. Scott and D. Rosenberg: Successful Robustness Analysis. Available at <http://www.drdoobs.com/184414712> [Online]. Last Accessed October 2010.
- [29] S. S. Shapiro and M. B. Wilk, "An Analysis of Variance Test for the Exponential Distribution," *Technometrics*, vol. 14, pp. 355-370, 1972.
- [30] S. Siegel, and N. J. Castellan Jr., *Non-parametric Statistics for the Behavioral Sciences* (2nd Edition). McGraw-Hill, 1988.
- [31] Visual Paradigm, "Visual Paradigm", [Online www.visual-paradigm.com]. Last November 2011.
- [32] J. L. Whitten and L. D. Bentley: *Systems Analysis and Design Methods*. 7th Edition. McGraw-Hill/Irwin. 2007.
- [33] C. Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell, and A. Wesslen, *Experimentation in Software Engineering - An Introduction*. Kluwer, 2000.



Mohamed El-Attar received his B.Eng. degree from Carleton University, Canada, and his Ph.D. degree from the University of Alberta, Canada. In 2009, he joined the department of Information and Computer Science at King Fahd University of Petroleum and Minerals, Saudi Arabia, as an assistant professor.

His research interests include Requirements Engineering, in particular with UML and use cases, object-oriented analysis and design, model transformation and empirical studies. For information about his research see:

<http://faculty.kfupm.edu.sa/ICS/melattar/index.html>.



Mahmoud O. Elish is an assistant professor of Software Engineering in the Information and Computer Science Department at King Fahd University of Petroleum and Minerals, Saudi Arabia. He received the PhD degree in Computer Science from George Mason University, USA. His research interests include software metrics and measurement, software design, software maintenance,

empirical software engineering, and software quality predictive models.



Sajjad Mahmood Sajjad Mahmood received his PhD in computer science from La Trobe University, Melbourne, Australia. He is an assistant professor in the Information and Computer Science Department, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia. His research interests include software reuse,

component-based software engineering and software product lines.



James Miller received the B.Sc. and Ph.D. degrees in Computer Science from the University of Strathclyde, Scotland. During this period, he worked on the ESPRIT project GENEDIS on the production of a real-time stereovision system. Subsequently, he worked at the United Kingdom's National Electronic Research Initiative on Pattern

Recognition as a Principal Scientist, before returning to the University of Strathclyde to accept a lectureship, and subsequently a senior lectureship in Computer Science. Initially during this period his research interests were in Computer Vision, and he was a co-investigator on the ESPRIT 2 project VIDIMUS. Since 1993, his research interests have been in Software and Systems Engineering. In 2000, he joined the Department of Electrical and Computer Engineering at the University of Alberta as a full professor and in 2003 became an adjunct professor at the Department of Electrical and Computer Engineering at the University of Calgary. He is the principal investigator in a number of research projects that investigate software verification, validation and evaluation issues across various domains, including embedded, web-based and ubiquitous environments. He has published over one hundred refereed journal and conference papers on Software and Systems Engineering (see www.steam.ualberta.ca for details on recent directions); and recently served as the program co-chair for the IEEE International Symposium on Empirical Software Engineering and Measurement; and sits on the editorial board of the *Journal of Empirical Software Engineering*. He regularly appears in the *Journal of Systems and Software* survey of "top scholars". This survey ranks leading researchers by their output in leading journals over a 5-year period. In the most recent survey, he was ranked the ninth most productive researcher in the world.