VARIATION IN STUDENTS' CONCEPTIONS OF OBJECT-ORIENTED INFORMATION SYSTEM DEVELOPMENT

Ilona Box and Raymond Lister*

1. INTRODUCTION

CRUD (create, read, update, delete) analysis in object-oriented information system development (OOISD) is recommended as a means to improve the quality of the resulting system [1-9]. Box [10] went further by stating that it also improved students' learning of OOISD. She stated, based on anecdotal evidence, that "the earlier [students] can detect errors of omission the more likely they are to succeed at "good" analysis and design; the more confident they feel about learning and doing OOISD and the better their object thinking".

If OOISD students did attempt CRUD analysis in the manner presented by Box & Ferguson[9], early, during analysis, would Box's statement, in part or whole, be supported? In the first instance, would it be possible to detect the conceptions students have about OOISD by examining students' CRUD matrices? In this paper, we explore the question of what are the variations in the conceptions constituted in CRUD matrices that are an outcome of CRUD analysis after the development of high-level use cases and the initial class diagram. We use a phenomenographic research approach to constitute the variation in conceptions. Our results are categories describing what types of errors the students make in the CRUD matrices. Based on 57 student assignments, we identify three categories representing various types of errors made by students. The categories of description, the outcome space of the research, contribute to the ISD community by providing a taxonomy of errors to inform teaching practice of OOISD and gives modest conditional support to Box's statement.

1.1. Phenomenography

Phenomenography is a research approach focusing on the qualitatively different ways people experience, understand, perceive, or conceptualise a phenomenon[11]. The underpinning philosophy is that there are a limited number of qualitative ways of experiencing phenomena. Phenomenographers usually collect their data by recording and tran-

* Both authors, University of Technology, PO Box 123, Broadway, NSW 2007. Ilona Box, also University of Western Sydney, Locked Bag 1797, PENRITH SOUTH DC, NSW 1797. ilona@cit.uws.edu.au

scribing interviews with a small number of interviewees. The transcripts are analysed to identify one or more dimensions of variation; a dimension of variation is a set of categories somehow related, e.g. linearly or hierarchically, with a small number of categories in the set. Since phenomenographers wish to capture the variation in experiences, and not quantify the popularity of each experience (though this can be done in follow-up work), they can work with small numbers of interviewees.

Bruce[12] has presented a synopsis of phenomenography in information technology research. Booth[13] conducted the seminal phenomenographic work in computing, "what does it mean and what does it take to learn to program?". Also in 1992, Gerber, Buzer, Worth and Bruce asked educators and researchers of geographical information systems (GIS) their views and experience of GIS[12]. Academics', students', and practitioners' concepts of information systems have also been explored[14]. Cope's[15] later study identified students' different ways of seeing information systems, providing insights into how students' ways of seeing differs from the views of experts in the field. Bruce notes, "The differences identified are educationally critical". Other studies in the information technology discipline published in recent years include: Berglund[16] on the understanding of computer networking protocols; McDonald[17] on the nature and acquisition of algorithm understanding; Klaus and Gable[12] on the understanding by senior managers of knowledge management in the context of enterprise systems; Stewart and Klaus[12] on the experience of the business-IT relationship. Other areas of research underway are learning to program and learning about computer networking and data communications[12]; and how students, who take their first programming course, understand objects and classes[18].

These works are leading the research in computing using phenomenographic studies. Bruce[12] states education research like this is critical to the design of effective professional education, and of some importance to information technology educators.

In this paper, we report upon our own phenomenographic study, to investigate the qualitatively different understandings students have of OOISD as constituted in students' CRUD matrices.

2. METHOD

Our data came from one source, 57 assignments completed by students as partial fulfilment of the first OOISD subject in an undergraduate computing degree. The assignments are work not collected by students at the end of semester and represent the five grades awarded for this assessment task. The assignment was an optional assessment tasks. Students chose to do the assignment to try for a credit or distinction grade.

The assignment was based on a case study (Appendix A) and required the students to correct and complete 12 high-level use cases, draw a class diagram, and do a CRUD analysis among a number of other tasks. The students were explicitly instructed to do the CRUD analysis before finalising their high-level use cases and class diagram and to follow the software development method as described in Box & Ferguson [9]. Templates of the CRUD matrices, as discussed in Box & Ferguson, are shown in Appendix B. As well as the discussion provided in the text, the students received instruction and practice in the use of the CRUD matrices.

The CRUD matrices from the assignments were analysed in the phenomenographic style. Our focus for analysis was drawn to the errors the students had made in the CRUD matrices. The analysis was an iterative process. We did not begin with the categories; we formed the categories from what we found in the data. The types of errors were

identified and then the set of CRUD matrices from each assignment were placed into one category or between categories. The categories were revised. The placing of assignments and revision of categories was iterated until we reached a consensus of what were the categories and the relationships between the categories. We only added a category when we could identify CRUD matrices in support of that category. The outcome space is the qualitative description of each category. The qualitative descriptions are of the predominant types of errors in a category and descriptive exemplars of how that error manifested in the CRUD matrices in the assignments.

It is important to understand that a single assignment is not designated to a single category. Students naturally have several conceptions about OOISD, although they may understand some conceptions better than others. Therefore, one assignment may fall across more than one of the following categories.

3. RESULTS

From the data, we identified three categories. The categories were constituted as aggregations of the types of errors (or lack thereof) we found in the CRUD matrices the students created.

3.1. Fragmented and Unstructured Conceptions

The first category is fragmented and unstructured conceptions. Here, it is difficult to identify conceptions that are correct. There are many conceptions that are incorrect. Often, one or more conceptions are in contradiction or conflict with other conceptions. We regard this as the less powerful understanding of OOISD. The various types of errors found in the CRUD matrices and which constitute this category are:

1) Object-oriented principles (encapsulation, data abstraction, inheritance):
   a) Identifying a class that only represents a chunk of data in the system. For example, an abstract class called *Data* [Assignment 6], a persistent class called *Records* with the attribute *Unit Outlines* [Assignment 7], a persistent class called *Record* [Assignment 45].
   b) Including a class name as an attribute in another class. For example, in the CRUD attribute matrix for the class *Unit Outline*, *unitCoordinator* is listed as an attribute, *Outline-Coordinator* is listed in the CRUD Association matrix, and *Unit Coordinator* is listed in the CRUD class matrix [Assignment 8].
2) Persistence (identifying persistent and transient objects, and abstract classes):
   a) Classes at the same level or specialisation in an inheritance structure are indiscriminately identified as persistent, transient, or abstract. For example, the classes listed in the CRUD class matrix: *Unit Coordinator, Team leader, Administrator,* and *Head of School* (as specialisations of person) are identified as persistent, transient, abstract, and abstract respectively [Assignment 1].
   b) Classes with objects that come into existence as part of the new system are identified as persistent; classes with objects that exist before the new system is built are identified as abstract. For example the classes: *Discipline team, Student, Team Leader, Unit coordinator,* and *Unit* exist in other software systems and were identified as abstract. The classes: *Outline, Summative assessment,* and *Terms and choices* do not exist in other software systems, are considered "new" classes and are labelled persistent [Assignment 7].

c) Classes are not identified as persistent, transient or abstract. For example, the classes: *Unit Outline, Unit Coordinator, Team Leader, University,* and *Administrator* are listed in the CRUD class matrix and the "P" column is left blank [Assignment 8].

d) Classes that are roles or actors are classified as persistent; classes that represent things are classified as transient. For example, the classes: *Team leader* and *Coordinator* are typed as persistent; the classes: *Unit outline* and *Calculate summative assessment* are typed as transient [Assignment 3].

3) Traceability (among CRUD matrices, and between use cases or the class diagram and CRUD matrices):

a) The number of classes shown in the class diagram is not the same number listed in the CRUD class matrix. For example, the class diagram contains 11 classes and the CRUD class matrix lists only five classes, [Assignment 3], the class diagram contains seven classes and in the CRUD class matrix 49 classes are listed [Assignment 6].

b) The number of associations shown in the class diagram is not the same number listed in the CRUD association matrix. For example, the class diagram contains 10 associations and the CRUD association matrix lists only six [Assignment 1].

c) The names of the use cases in the CRUD matrices are not the same as in the use case diagram or high-level use cases. For example, the use cases: *Create unit outline, Add new team leader, Calculate summative assessment,* and *Record summative assessment,* are shown in the use case diagram and high-level use cases; in the CRUD class matrix the use cases are listed as: *A Unit outline, New team leader,* and *Calculate/record Summative assessment* [Assignment 16].

d) The identification of classes is inconsistent between matrices. For example, in the CRUD class matrix classes are identified as: *Unit Coordinator* and *Unit Outline,* and in the CRUD association matrix the classes are identified as: *Coordinator, Outline,* and *UnitOutline* [Assignment 8].

e) The attributes in the class in the class diagram are not the attributes listed in the CRUD attribute matrix for the class. For example, the class *Unit outline* has the attributes: *coarse title, course ID,* and *semester* in the class diagram; in the CRUD attribute matrix the attributes listed are: *Name, Idnumber, Address,* and *Phone number* [Assignment 3].

f) Only some of the attributes in the class diagram for a class are listed in the CRUD attribute matrix for the class. For example, the class *Unit Outline* in the class diagram has the attributes: *Title Page, Main Page,* and *Summative Assessment*; in the CRUD attribute matrix for the class the attributes are: *Titlepage, Main Heading, Unit Number, Unit Name,* and *Summative assessment* [Assignment 4].

g) The use cases identified in the CRUD class matrix to create, read, update, or destroy are not listed in the CRUD attribute matrix where they would be expected. For example, in the CRUD class matrix, the class *Unit Outline* is created during the use case *Create Unit Outline*; in the CRUD attribute matrix for the class *Unit Outline* all the attributes are initialised at creation by the use case *Define Outline* [Assignment 8].

4) Notation conventions (the notation of class, association, attribute, and use case names):

a) Little or no consistency between the notation for the names of classes, attributes and/or use cases between the class diagram, use case diagram, or high-level use

cases, and the CRUD matrices. For example, five out of six class names in the class diagram has each word in title case, and in the CRUD class matrix, the four classes listed are written in sentence case [Assignment 21].

b) Little or no conformity to the notation for the names of classes, attributes, and/or use cases stipulated in the software process. For example, in the process described by Box & Ferguson, which conforms to UML version 1.4, class names are written in upper camel case (e.g. UnitOutline), attribute names are written in lower camel case (e.g. unitNumber), and use cases are written in sentence case (e.g. Create unit outline).

5) Process (the functions the system needs to perform):

a) For the classes that can be matched to people, a description or N/A rather than the use case is entered in the CRUD class matrix. For example, in the CRUD class matrix, the class *Unit Coordinator* has for create *Unit Outline*, has for read *N/A*, has for update *Make changes in unit outline*, and has for delete *Delete data from unit outline*; the class *Team Leader* has for create *N/A*, has for read *Read the unit outline create by unit coordinator*, has for update *Approval unit outline and update it*, and has for delete *Send data to unit coordinator for delete data from unit outline* [Assignment 17].

6) Design (the representation of design decisions relating to the identification of classes, associations, attributes, and use cases in CRUD matrices):

a) Identifying more than one use case that performs the same function. For example, objects in the class *Unit Outline* are read during the use case *Read unit outline*; objects in the class *Archive* are read during the use case *Access unit outline* [Assignment 1].

b) Classes and use cases identified using the same nomenclature. For example, in the CRUD class matrix the class is named *Calculate summative assessment* and the use case to create objects in this class is named *Calculate assessment* [Assignment 3], in the CRUD association matrix the associations are identified as *Approve unit outline, Modify unit outline,* and *Submit unit outline* [Assignment 21].

c) The attributes do not belong to objects in the class in a CRUD attribute matrix. For example, the class *unit outline* has the attributes: *Name, Idnumber, Address*, and *Phone number* [Assignment 3].

d) Identifying and naming individual parts in whole-part associations when the parts have common attributes. For example, listing in the CRUD class matrix the classes *AssumedKnowledge, ClassHours, Component, Content, Cover, CoverPage, Disabilities, Exclusion, Introduction, LearningSkillsUnit, Malpractice, Method, Note, OutOfClassHours, Practice, Prerequisite, Presentation, RecommendedText, References, StaticNote, StudentLearningOutcome,* and *SummativeAssessment* as classes that have objects created during the *Create unit outline* use case [Assignment 6].

e) Identifying separate classes for the same data. For example, *Outline* and *Records* [Assignment 7], *Unit Outline* and *Archive* [Assignment 5].

f) Classes and associations are identified as separate due to the timing or sequence of events. For example, the class *Unit outline* is created during the *Create unit outline* use case, the class *Approval* is created during the *Submit unit outline* use case, and the *Approval-ApprovalResult* association is created during the *Approve unit outline* use case [Assignment 15].

3.2. Conceptions About Process Override Conceptions About Objects

Here, conceptions about what the system needs to do override conceptions about object-orientation. Where the processing of the system and the CRUD analysis are aligned the CRUD matrices are correct. There are fewer errors than in the previous category. The various types of errors (or lack thereof) found in the CRUD matrices and which constitute this category are:

1) Object-oriented principles (encapsulation, data abstraction, inheritance):
   a) The majority of needed classes are listed, however, the understanding of object-oriented principles lacks the awareness to separate data appropriately. For example, the class *UnitOutline* is not separated into classes of *Unit* and *UnitOutline* [Assignment 2], the class *Unit Outline* is duplicated as the class *Template* [Assignment 13], the class *Unit* is listed, though struck out, in the CRUD class matrix and does not appear in the class diagram [Assignment 46].
   b) There is little, if any, use of inheritance.
2) Persistence (identifying persistent and transient objects, and abstract classes):
   a) The class that is a role or actor that could be seen as outside the system is classified as abstract; all other classes are correctly classified as persistent. For example, only the class *Administrator* is typed as abstract [Assignment 43], only the class *User* is identified as abstract [Assignment 46].
   b) Incorrect identification of persistence is rare.
3) Traceability (among CRUD matrices, and between use cases or the class diagram and CRUD matrices):
   a) The trace errors are minor inconsistencies. For example, for the minority of classes in the class diagram the order of the attributes is not the same as the order of the attributes in the corresponding CRUD attribute matrix for these classes [Assignment 2], the class in the class diagram is named *Explanations* and in the CRUD class matrix is listed as *Explanation* [Assignment 13].
4) Notation conventions (the notation of class, association, attribute, and use case names):
   a) Consistency between the notation for the names of classes, attributes and/or use cases between the class diagram, use case diagram, or high-level use cases, and the CRUD matrices is good though some errors still occur. For example, the use of abbreviations in the use case name, such as, *Calculate sum. assess. total* [Assignment 42].
   b) Conformity to the notation for the names of classes, attributes and/or use cases stipulated in the software process is good though some errors still occur. For example, attribute names are written in upper camel case [Assignment 2], class names and attribute names include spaces between words [Assignment 13].
5) Process (the functions the system needs to perform):
   a) For classes representing actors, the use cases chosen to complete the CRUD class matrix are those with which the actors are associated in the use case diagram. For example, the class *Administrator* has for create, read, update, and destroy the use case *Administer users* [Assignment 2].
   b) The class is being considered in terms of its part in the process rather than the objects belonging to the class. For example, the class *Team Leader* has for create *Add new unit coordinator*, and for read *Approve unit outline* [Assignment 37].

c) The identification of the class is a step in the process rather than one to which objects would belong. For example, the classes *Approval* and *Submit* [Assignment 2], *Calculation* and *Submission* [Assignment 43], *Report* [Assignment 47], and *Submission* [Assignment 50].

6) Design (the representation of design decisions relating to the identification of classes, associations, attributes, and use cases in CRUD matrices):
   a) The choice of a particular use case is not in keeping with many of the other use cases chosen in the CRUD matrices. For example, the class *Summative assessment* has for create *Calculate sum. assess. total*, and has for read *Create unit outline*, the class *Unit Outline* has for create *Create unit outline*, and has for read *Approve unit outline* [Assignment 42].
   b) Only a few classes are identified beyond the initial, easily identified set of classes. For example, the initial, easily identified, set of classes includes: Administrator, TeamLeader, UnitCoordinator, UnitOutline, DisciplinedTeam, and Summative-Assessment, beyond this set the classes listed in CRUD matrices are: Help, School, School Archive [Assignment 46], Explanatory Document [Assignment 50], Unit, Unit Offering, and Campus [Assignment 19].

3.3. Conceptions of Design Decisions Appropriate within the Object-oriented Paradigm

This category shows the more powerful understanding of OOISD. The types of errors are predominantly about incorrect or weak design choices. The various types of errors (or lack thereof) found in the CRUD matrices and which constitute this category are:

1) Object-oriented principles (encapsulation, data abstraction, inheritance):
   a) There is more and accurate use of inheritance. For example, the classes *FormativeAssessment* and *SummativeAssessment* inherit from the class *Assessmen*t, and the classes *Administrator, TeamLeader,* and *UnitCoordinator* inherited from the class *Person* [Assignment 24].
2) Persistence (identifying persistent and transient objects, and abstract classes):
   a) Incorrect identification of persistence is rare. For example, the classes *Assessment* and *Person* are correctly, and the only classes, identified as abstract classes [Assignment 24].
3) Traceability (among CRUD matrices, and between use cases or the class diagram and CRUD matrices):
   a) Trace errors do not occur.
4) Notation conventions (the notation of class, association, attribute, and use case names):
   a) Consistency between the notation for the names of classes, attributes and/or use cases between the class diagram, use case diagram, or high-level use cases, and the CRUD matrices is good though some errors still occur. For example, the use of abbreviations in the use case name, such as, *Calculate sum. assess. total* [Assignment 42].
   b) Conformity to the notation for the names of classes, attributes and/or use cases stipulated in the software process is good though some errors still occur. For example, attribute names are written in upper camel case [Assignment 2], class names and attribute names include spaces between words [Assignment 13].

5) Process (the functions the system needs to perform):
   a) The object-oriented paradigm is at the fore. The processes performed by the system are presented as use cases that determine the creation, reading, updating, and destruction of objects within classes.
6) Design (the representation of design decisions relating to the identification of classes, associations, attributes, and use cases in CRUD matrices):
   a) Indications of consideration of the consequences of making design decisions. For example, the notes: *Should TeamLeader and UnitCoordinator objects be destroyed, if they contain information on which session they apply to?* [Assignment 24] and *Summative assessment is part of the unit outline and cannot be destroyed* [Assignment 9].
   b) An awareness that use cases have a limit to the functions for which each is responsible. For example, the case study provided 12 use cases; in a CRUD class matrix listing 19 classes, seven Create/Class cells, two Read/Class cells, 10 Update/Class cells, and 14 Destroy/Class cells were not assigned one of the 12 provided use cases. [Assignment 36].
   c) The identification of a singleton object. For example, the note *A single Unit Outline Handler instance must be created when the system is installed... should never be destroyed* [Assignment 10].
   d) Objects that are associated as actors to a use case need be read. For example, objects in the class *Team Leader* are read during the use case *Approve outline* [Assignment 12].

4. DISCUSSION

In considering these results, we need to keep in mind two mistakes that can arise from a misunderstanding of phenomenography. First, phenomenography is a qualitative method of research, not quantitative. Hence we draw no conclusions about the number of assignments that fall into any of the above categories or the frequency with which students fit into a category. To make such conclusions would require significantly more data and a different research approach. The aim of phenomenographic research is to capture diversity. Second, the categories do not represent a single assignment. Typically, if an individual is shown the categories generated from phenomenographic research, they will identify with more than one position. There may be some positions to which they identify very strongly, and some positions to which they do not identify at all, but it is rare for a person to identify with only one category.

The assignments provided 57 separate sources of data, which is a relatively high number of sources for a phenomenographic study. Phenomenographers often continue to collect data until they believe they have reached “saturation”. That is, they collect data and analyse it concurrently, ceasing to collect data when they have several consecutive interviews that do not lead to the identification of new categories. From our 57 sources, we do not claim to have reached saturation because the assignments were those not collected by students even though the assignments span the range of grades awarded. However, it was felt that within this data set the categories are a reasonable outcome space, i.e. categories of description of the variation in students’ conceptions of OOISD constituted in aggregations of the types of errors or lack thereof made in CRUD matrices.

Examining more CRUD matrices in more assignments may add more categories, but is unlikely to invalidate the categories we have identified in this paper. Students

chose to do the assignment to try for a credit or distinction grade. All students were required to attempt a final exam of 60 multiple-choice questions for a pass grade. The students whose assignments were used as the data set for this phenomenographic study received scores for the final exam ranging from 20 to 46, where 19 was the lowest score and 52 the highest for all students. The assignments are therefore a reasonable representation of the diversity of CRUD matrices in assignments. However, the study could benefit from follow-up work such as interviews where students are questioned about their understandings while undergoing the experience of doing a CRUD analysis.

Phenomenographers do not necessarily identify a unique set of categories from the same data. For example, if Cope [15] examined our data set, he may find evidence for the same categories he identified in his study of students' different ways of seeing information systems. Or if Eckerdal [18] were to examine our data set, she may find evidence for the same categories she identified in her first major study of students understanding of the concepts object and class. The categories identified in any study are to some extent dependent on the intent of the phenomenographer. Our intent was to identify the types of errors students made, and we chose our categories accordingly.

If phenomenographers do not necessarily identify a unique set of categories from the same data, is phenomenographic work therefore reliable and valid? Phenomenographic work can be considered valid and reliable in the following sense. If two people were given the description of the focus of the study, some categories, and some quotes from data, those people would usually place the quotes into the same categories. The readers can determine for themselves whether they would place most of the above examples into the same categories as those into which the authors have placed them.

In constituting our categories, we wanted to focus on the CRUD matrices in an assignment that also contained high-level use case descriptions, a use case diagram, and a class diagram. An analysis of these models could reveal different categories or dimensions of variation in students' conceptions of OOISD. We acknowledge this, but we regard it as separate to our concern. By focusing on the CRUD matrices and limiting our consideration of the other models to their relationship with the CRUD matrices, we identified a taxonomy of errors to inform teaching practice of OOISD. If the way OOISD is taught can be informed by this study, then a taxonomy of errors adds value to the Box & Ferguson CRUD analysis method.

For one of the authors, an unexpected insight to emerge from this study is the potential for CRUD analysis to be used as the first instrument for teaching OOISD. Until this study, the author had not intuited or reasoned about this possibility. We believe that it is possible to identify students' conceptions of OOISD by providing OO analysis models, asking the students to complete a CRUD analysis, and then design interventions to correct the students' conceptions of OOISD. This falls in line with the arguments of teaching students to read program code before writing program code. We would be asking students to read OO analysis models, complete CRUD matrices as a demonstration of their understanding of the models, before asking them to create OO analysis models.

5. CONCLUSIONS

It was asserted that CRUD matrices improve students learning of OOISD (Box, 2003). We doubt that CRUD matrices, of themselves, can improve students learning of OOISD. However, students do have varying conceptions about OOISD. We conducted a phenomenographic study that identified three categories of students conceptions about

OOISD constituted as aggregations of types of errors or lack thereof made in CRUD matrices. From an analysis of 57 data sources, the categories are: fragmented and unstructured conceptions, conceptions about process override conceptions about objects, conceptions of design decisions appropriate within the object-oriented paradigm. These categories are probably not an exhaustive list, but we believe that further data gathering will not invalidate these categories.

These categories can be used to inform the teacher of students' affinities with the categories, from strongest to weakest. We believe that the teaching and learning of OOISD can be improved with the early use of CRUD matrices and at the same time informing students of the taxonomy of errors. Then introducing interventions to help make the students conceptions of OOISD more powerful. Our hope is that educators will use the categories we have identified to make explicit to students the errors that are known to occur if their understanding of OOISD is weak.

Beyond CRUD matrices, this paper demonstrates how phenomenography can be used as a tool for constituting categories of students' understandings of ISD. It can be used to define least powerful to most powerful understandings, before deciding on subject design or during the formative evaluation of subject design. We found the effort of analysing our data led to a reflection, or summative evaluation, of the subject design. By the time we finished the analysis, we saw merit in the information revealed in the categories, not just the more powerful category to which we had ascribed the most value. Indeed, we found variations of which we were not fully aware prior to this study. Beginning with a phenomenographic study may therefore lead to a more comprehensive approach to subject design in general.

REFERENCES

1. Gottesdiener, E., *OO Methodologies: Process and Product Patterns.* Component Strategies, 1998. **1**(5).
2. Satzinger, J., R. Jackson, and S. Burd, *Systems Analysis and Design in a Changing World*. 2nd ed. 2002, Boston, MA, USA: Course Technology.
3. Maciaszek, L.A., *Requirements Analysis and System Design: Developing Information Systems with UML*. 2001, Harlow, England: Addison-Wesley.
4. Brown, D.W., *An Introduction to Object-Oriented Analysis, Objects, and UML in Plain English*. 2nd ed. 2002: John Wiley & Sons.
5. Dennis, A., B.H. Wixom, and D. Tegarden, *Systems Analysis and Design: An Object-oriented Approach with UML*. 2002: John Wiley & Sons, Inc.
6. Brandon, D., Jr., *CRUD matrices for detailed object oriented design.* The Journal of Computing in Small Colleges, 2002. **18**(2): p. 306 - 322.
7. Armour, F. and G. Miller, *Advanced Use Case Modelling: Software Systems*. 2001: Addison-Wesley.
8. AMS, *AMS best practices use cases: Advanced use case modeling*. 2003, AMS.
9. Box, I. and J. Ferguson, *Object Oriented Software Development: Step by Step*. 2002, Sydney: Pearson Education.
10. Box, I., *Old Trick, New Dogs: Learning to use CRUD matrices early in object-oriented information system development*, in *Constructing the Infrastructure for the Knowledge Economy: Methods & Tools, Theory & Practice*, Twelfth International Conference On Information Systems Development, Editor. 2003, Kluwer: Melbourne.
11. Marton, F., *Phenomenography-a research approach to investigating different understandings of reality*. Journal of Thought, 1986. **21**(3): p. 28-49.
12. Bruce, C. *Phenomenography in the Centre for Information Technology Innovation (CITI)*. in *Current Issues in Phenomenography Symposium*. 2002. Canberra, Australia.
13. Booth, S., *Learning to program: A phenomenographic perspective (Dissertation abstract)*. 1992.
14. Cope, C., P. Horan, and M. Garner, *Conceptions of an Information System and Their Use in Teaching about IS*. Informing Science, 1997. **1**(1): p. 9-22.
15. Cope, C., *Educationally critical aspects of the experience of learning about the concept of information systems*. 2000, La Trobe University: Melbourne.

16. Berglund, A., *On the Understanding of Computer Network Protocols*, in *Department of Information Technology*. 2002, Uppsala University: Uppsala. p. 76.
17. McDonald, P. *The nature and acquisition of algorithm understanding: a phenomenographic investigation.* in *Current Issues in Phenomenography Symposium*. 2002. Canberra, Australia.
18. Eckerdal, A., *Resources for Learning Object-oriented Programming*. 2003, SIGCSE 2003 Doctoral Consortium.

APPENDIX A: UNIT OUTLINE CREATION AND APPROVAL CASE STUDY

Students were provided with a detailed description of the following case study and the assignment requirements, which included CRUD matrices as described by Box and Ferguson[9]. (A unit is synonymous with subject, course, or paper.)

This study is about the creation and approval of unit outlines within a school of computing and IT (SCIT). The system includes the creation of outlines by unit coordinators, the approval of the outlines by team leaders, and the creation of reports by administration staff.

Currently, the unit coordinator of each unit writes the unit outline, usually based on the previous version, rarely from scratch. After a team review, the team leader approves the outline, indicated by signing the cover page, and forwards it to the administration. The outline is made available online and archived in the SCIT's records. The current system is segregated and requires too much manual intervention. The desired system should reduce the time to enter and maintain outlines and derive reports.

An outline needs to be created each time a unit is delivered and to standards specified by the school and university. All outlines must have a consistent layout. An outline contains information that is the same for all outlines for a semester and information that varies from unit to unit, such as the content and learning objectives. The outline will start with a title page and contain sections such as: prerequisites, exclusions, assumed knowledge, introduction, student learning outcomes, content, delivery mode, practices of the school concerning assessment, method of assessment, clauses regarding academic malpractice, disabilities, and the learning skills unit, the recommended text and readings, unit coordinator, lecturer, and tutor contact details. The user interface needs to behave in such a way, as much as possible, so the coordinator can select options. The following limited set of high-level use cases were provided for correction and completion by the students.

Use case: Approve unit outline
Category: Core
Actors: Team Leader
Trace: <enter the traces to the business functions>
Description: This use case begins when a Team Leader chooses to approve a unit outline. The authority of the Team Leader is verified and the unit outline is marked as approved. On completion, the team leader and unit coordinator are notified of the approval, the unit outline is made available to the students and a copy is kept in the school archive.

Use case: Calculate summative assessment total
Category: Core
Actors: Team Leader
Trace: <enter the traces to the business functions> Included by: Record summative assessment
Description: This use case begins when a unit coordinator has completed the entry of all summative assessment. The summative assessment total is calculated by adding together the value of each summative assessment. On completion, the summative assessment total is shown onscreen.
Notes: The summative assessment total must equal 100.

Use case: Record summative assessment
Category: Core
Actors: Team Leader
Trace: <enter the traces to the business functions> Includes: Calculate summative assessment total
Description: This use case begins when a unit coordinator commences the entry of a summative assessment. The summative assessment details are recorded. On completion, the unit coordinator verifies that all summative assessment is complete.

Use case: Add new unit coordinator
Category: Core
Actors:
Trace: <enter the traces to the business functions> Extend from: Create unit outline, Administer users
Description: This use case begins when…. … The use case ends when… .

Use case: Modify existing unit coordinator
Category: Core
Actors:
Trace: <enter the traces to the business functions> Extend from: Create unit outline, Administer users
Description: This use case begins when…. … The use case ends when… .

Use case: Create unit outline
Category: Core

Actors:
Trace: <enter the traces to the business functions> Includes: Record summative assessment
Description: This use case begins when…. … The use case ends when… .

Use case: Provide explanation
Category:
Actors:
Trace: <enter the traces to the business functions> Extend from: Create unit outline
Description: This use case begins when…. … The use case ends when… .
Notes: Explanations of terminology and choices available when creating the unit outline.

Use case: Add new team leader
Category:
Actors:
Trace: <enter the traces to the business functions> Extend from: Administer users
Description: This use case begins when…. … The use case ends when… .

Use case: Modify existing team leader
Category:
Actors:
Trace: <enter the traces to the business functions> Extend from: Administer users
Description: This use case begins when…. … The use case ends when… .

Use case: Administer users
Category: Core
Actors: Administrator
Trace: <enter the traces to the business functions> Extends to:
Description: This use case begins when…. … The use case ends when… .

Use case: Modify unit outline
Category:
Actors:
Trace: <enter the traces to the business functions>
Description: This use case begins when…. … The use case ends when… .

Use case: Submit unit outline
Category: Core
Actors:
Trace: <enter the traces to the business functions> Extend from: Create unit outline, Administer users
Description: This use case begins when…. … The use case ends when… .
Notes: A unit coordinator must indicate that the unit outline is ready for the team leader's approval.

APPENDIX B: TEMPLATES FOR THE CRUD MATRICES

The following templates for the CRUD matrices are reproduced from Box and Ferguson (2002, p. 122, 123 & 124)

CRUD Class Analysis

Class	**P**	**Create**	**Read** (utilise)	**Update** (maintain)	**Destroy**
ClassName	*P/T/A*	*Use case name*	*Use case name*	*Use case name*	*Use case name*

Notes: _____

Figure 1. CRUD class matrix format from Box & Ferguson, 2002, p. 122.

CRUD Association Analysis

Association	**Create**	**Read** (utilise)	**Update** (maintain)	**Destroy**
ClassAName-ClassBName	*Use case name*	*Use case name*	*Use case name*	*Use case name*

Notes: _____

Figure 2. CRUD association matrix format from Box and Ferguson, 2002, p. 123.

CRUD Attribute Analysis – Class: *ClassName*

Attribute	**Initialise at Create**	**Read** (utilise)	**Update** (maintain)	**Reset**
attributeName	*Use case name*	*Use case name*	*Use case name*	*Use case name*

Notes: _____

Figure 3. CRUD attribute matrix format from Box & Ferguson, 2002, p. 124.