Evaluating the Semantic Similarity of Object-Oriented Domain Models: Achieving Greater Predictability by Performing Behavioral Analysis First

Davor Svetinovic Daniel M. Berry Michael W. Godfrey

Nancy A. Day

David R. Cheriton School of Computer Science

University of Waterloo

Waterloo, Ontario, Canada

{dsvetino, dberry, migod, nday}@uwaterloo.ca

Abstract

The main goal of any object-oriented analysis (OOA) method is to clarify a problem by modeling the problem and its domain. Therefore, the most important artifact that results from OOA is the domain model*, which is usually realized as a class diagram that describes the core concepts in the domain and their relationships. Ideally, a mature engineering process is repeatable: analysts given the same problem and instructions to follow the same OOA process should produce semantically similar domain models.*

This work compares the observed semantic similarity among the different domain models produced by one process for one system by different users of the process when the process is one

of:

1. *creation of use cases (UCs), then sequence diagrams, then a domain model, and*

2. *creation of UCs, then a unified UC statechart, then a domain model.*

One process was used to produce 31 specifications of a large VoIP system and its accompanying information management system. The other process was used to produce 34 specifications of the same system. The data show that domain models produced using the second process were 10% more semantically similar to each other than those produced using the first process, but at a cost, by one measure, of up to 25% more time, spent in learning the process and in requirements elicitation.

1 Introduction

In 1967, Dahl and Nygaard presented the first object-oriented programming (OOP) language, Simula 67 [8]. In 1982, Booch published his paper on object-oriented design (OOD) [4]. In 1988, Shlaer and Mellor published their book on object-oriented analysis (OOA) [27]. These three events were major milestones in the development of the object-oriented (OO) paradigm of software development. Today, object orientation is not just one of the oldest software development paradigms, it is also one of the most widespread. From OOP languages to different OO modeling standards and frameworks, object orientation shapes the ways we think about business and software systems, how we organize our development processes, and so on.

From more than 25 years of hindsight, it appears that the eventual widespread adoption of object orientation was fueled partially by the impact of Booch's 1982 paper "Object-Oriented Design" [4], and in particular, due to his claim of how easy it is to identify the objects of a problem, their attributes, and their operations by looking for nouns, adjectives, verbs, and adverbs in a written description of the problem. This identification forms the essence of an OOD method that Booch was advocating. In turn, these two steps form the foundation of what is today known as OOA.

With almost equal hindsight, we can safely say that these tasks are *not that simple* and, as stated, they are not sufficient for the production of high-quality OOA models. Indeed, Hatton [14] has presented empirical evidence that calls into question the fundamental idea that the main benefit of object orientation is that an object-oriented program accurately models its domain, and therefore its validity is easy to ascertain. Earlier, Santos (now Ramos) and Carvalho [25] performed an empirical assessment of the applicability of OOA to the development of information systems (ISs) and found that with their subjects, OOA led to good models of an IS itself but did not lead to adequate models of the processes of the organization that owns the IS. More generally, Hatton [14], Kaindl [16], and Kramer [17] have indicated an urgent need for experimentation aimed at validating the effectiveness of all software engineering abstraction techniques and methods including object-oriented techniques.

This paper and a companion paper [30] are derived from the first author's Ph.D. thesis [29], which describes an attempt to fulfill the need identified by Hatton, Kaindl, and Kramer with respect to one particular OOA method, the Use Case Unification Method (UCUM), a method to build a domain model (DM) of a computer-based system (CBS) that is to be developed. In the OOA literature [e.g., 20], this DM is known as a OOA class model or as a conceptual model.

This DM is built in the context of an RE effort to elicit and analyze requirements, and eventually to specify in a Software Requirements Specification (SRS) document the CBS's desired behavior and properties, i.e., the CBS's *requirements*. The portion of the real world that a CBS is supposed to automate is the CBS's *domain*. In use-case-driven requirements analysis methods [e.g., 21], the first task analysts perform in modeling the behavior of the CBS being built is to write *use cases* (UCs) that describe the CBS's intended behavior. A UC of a CBS is one particular way some user of the CBS uses the CBS to achieve stakeholders' goals. Domain experts and analysts together typically capture UCs during and after requirements elicitation from many stakeholders, each with a different perspective. The description of a UC is typically given at the shared-interface level, showing the CBS as a monolithic black box.

From these UCs, the analysts begin to model the entire CBS's domain. In object-oriented anal-

ysis (OOA), the analysts break down and describe the entire CBS's domain in terms of objects that are used later as the main source of objects for object-oriented design (OOD). *Conceptual analysis* is this whole process of discovering and specifying concepts from a domain with the goal of producing a DM. Traditionally, conceptual analysis proceeds by creating sequence diagrams from the UCs, then choosing objects, and finally assigning behavior to these objects in order to capture the combined behavior of all UCs.

Recently, among others, Glinz [11]; Whittle and Schumann [31]; and Harel, Kugler, and Pnueli [13] have suggested in three different formal methods that it might be worthwhile to build a behavioral model of the entire domain prior to decomposing the DM into objects. UCUM, as an implementation of this suggestion, was developed iteratively and collaboratively by us and some of our students as a means to deal with difficulties that our students were having when they tried to build DMs for CBSs in courses we were teaching. In UCUM, a statechart [12] representation of the CBS's domain as a UC statechart is constructed prior to decomposing the system into objects. Thus, UCUM is an informal variant of the more formal methods that suggested its creation. UCUM is described in detail both in the first author's Ph.D. thesis and in the companion paper, which shows an example application of UCUM to build a DM for a standard Turnstile Control System [3].

The companion paper discusses the difficulties we had observed among our students, confirming that OOA is not as simple as the folklore claims it is. It presents a qualitative evaluation of UCUM's effectiveness in attacking these difficulties. The companion paper concludes that UCUM is straightforward to learn and to apply, that it provides signifcant help in handling the observed DM building difficulties, that it helps students produce better than average models, but that it is not the cure to all DM building difficulties.

The present paper addresses a different measure of the quality UCUM as a method. It compares the repeatability of UCUM to that of traditional OOA. Ideally, given the same problem domain and the same OOA method, identical DMs should result. However, achieving identical DMs is highly unlikely in practice: different analysts may have different abilities and experiences, and these

differences often strongly influence the results of performing domain analysis. Hence, we settle for semantically *similar* DMs as a sign of a repeatable method for generating DMs. Therefore, the repeatability of a method is assessed by examining the semantic similarity of the DMs that result from different people using the method on the same problem in the same context or environment. The results from the data we gathered show that the DMs of one large CBS produced by students using UCUM are about 10% more semantically similar to each other than are the DMs of the same large CBS produced by students using more traditional OOA with no notion of unification of use cases.

In addition to the quantitative comparison of the repeatability of two OOA methods, this paper provides also (1) a discussion of the role of semantic similarity in evaluating a method's repeatability; (2) for the sake of experimental replication, a detailed description of how to determine in which of two sets of DMs are the DMs more semantically similar to each other; and (3) an in-depth analysis of the quantitative results, which provides insights for further improvements of OOA methods.

Accordingly, Section 2 discusses repeatability and explains why it is a desirable property of methods. Section 3 describes the background and context of the two case studies, each with a set of DMs produced with the help of one of the OOA methods. Section 4 describes a procedure for comparing the semantic similarity of the DMs in two sets of DMs. Section 5 describes the actual comparison of the semantic similarity of the elements of each of the two sets of DMs that are in the two case studies. Section 6 describes the results and the lessons learned from the case studies. Section 7 discusses related work whose results conflict with those of this paper. Section 8 concludes the paper.

2 Repeatability and Semantic Similarity

The goal of this paper is to determine whether UCUM as a method of building DMs is more repeatable than is traditional OOA with no notion of unification of use cases. Given two methods, M and N, for building DMs, M is deemed more repeatable than N if the DMs of a CBS produced

by analysts using M are more *semantically similar* to each other than are the DMs of the same CBS produced by similar analysts using N. By "similar analysts" we mean designers of roughly the same background and experience, such as students taking the same course at one university.

2.1 High Level Definition of Semantic Similarity

We use the term *semantic similarity* to indicate the degree of closeness that independently produced DMs of one domain may share. More formally, we use a generalized definition of semantic similarity [2]:

> Semantic similarity, variously also called semantic closeness/proximity/nearness, is a concept whereby a set of documents or terms within term lists are assigned a metric based on the likeness of their meaning/semantic content.

The metric used in this work is domain-expert opinion combined with manual clustering. The actual method used to evaluate semantic similarity is described in Section 4 after the presentation of data from the case studies that prompted the discovery of the evaluation method. For now, this section assumes an intuitive notion of semantic similarity.

Semantic similarity is not new and has been used quite extensively in practice and research. For example, the basis for the concepts of reference architectures and design patterns [e.g., 10, 7] is in the idea that all architectures matching a reference architecture and all design patterns matching a particular design are semantically similar.

2.2 Why Repeatability?

Each of science and engineering depends on repeatable results. Science requires that a result be reproducible before it can be accepted as fact, and engineering strives to develop reliable processes for achieving consistent outcomes. Furthermore, there has recently been much discussion of the scientific underpinnings of modelling; Booch, for example, has discussed the notion of modelling-as-science being something worth striving for [5].

In the context of software RE, an ideal modeling process is one that produces high quality models consistently and reliably. That is, for any given problem and encompassing domain, two teams of requirements engineers who follow the process are likely to come up with semantically similar solutions that are of good quality.

Repeatability for OOA as an engineering method means that given the same domain, different analysts applying the same OOA would be expected to produce semantically similar models of the same domain. Thus, semantic similarity of the results of several applications of an OOA method to one domain is a direct measure of the repeatability of the OOA method.

2.3 Semantic Similarity, Method Repeatability, and Quality

Guttorm Sindre, one of the authors of "Understanding Quality in Conceptual Modeling" [22] and of other works dealing with the quality of RE modeling [18, 19], observed in paraphrased private communication [28], that

- Semantic similarity of models does not prove quality, as the models could be of similarly poor quality. There are many factors other than just pure modeling that affect the similarity and the quality of models. However, it would be reasonable to assume that the fact that two independently produced models are similar at least increases the confidence that both modeling efforts have been performed well and that this confidence grows as the number of independently produced models grows beyond two.

- Semantic dissimilarity of models need not imply poor quality of any of the models, as the models could be for different purposes and from different viewpoints. However, in a situation in which (1) all analysts start from the same and very fixed source of information, and each is asked to produce a model with the same and very fixed purpose—as is often done in student exercises or experiments—and in which (2) the semantic quality of the models is measured as the correspondence between the model and the textual description, one might expect more semantic similarity between independently produced models than in other situations. In this case, it might be possible to argue that semantic similarity of the models increases the

likelihood that all are of high quality. In this case, semantic similarity might be seen as a sign of quality not only of the models and modelers but also of the modeling language and modeling method; that is, the models are more semantically similar because the modeling method is clear and easily followed, supporting consistent application of modeling language concepts by independent modelers.

3 Background and Context

The motivation for the development of UCUM came from our observations of our students' work in the requirements analysis and specification of a computer-based system (CBS) composed of

1. a telephone exchange or a Voice-over-IP (VOIP) system and
2. its information management system (IMS).

Production of the specification, in the form of a SRS document, is the term-long project carried out in the first course of a three-course sequence of software engineering courses that span the last three terms of the undergraduate software engineering program at the University of Waterloo [26]. In later courses, students design, implement, test, and enhance the CBS specified in the SRS.

From 2000 until 2005, the first author played a wide variety of roles in this course, serving as teaching assistant (TA), group coordinator, UML and SDL instructor, project evaluator, and finally, head TA. The first author reviewed over 135 SRSs, out of over 195 that were developed in this time interval by 3-or-4-student groups of over 740 software engineering, computer science, and electrical and computer engineering students. During the same period, each of the other authors was the instructor-in-charge for at least one offering of the course. The instructor-in-charge has overall responsibility for the contents, requirements, and marking for the course. This educational experience has given the authors the opportunity to observe various software analysis and specification issues from different perspectives.

The project in the three-course sequence involves using various techniques for developing software for real-time systems and OO techniques for developing information systems. Use cases (UCs) [e.g., 20] are used to capture requirements, and OOA is used as a bridge to later OOD. The real-time components of the CBS are specified using formal finite-state modeling in the Specification and Description Language (SDL) [6]. The information-system components of the CBS are specified using the notations of the Unified Modeling Language (UML) [24]. In addition, students are responsible for modeling user interfaces of the IMS and for the overall management of the requirements specification process. The average size of the resulting SRS document for the whole CBS is 120 pages, with actual sizes ranging anywhere from 80 to 250 pages.

In order to deal with the difficulties that the students had when using UC-driven OOA, we decided to introduce a method based on performing detailed behavioral analysis of the domain through the unification of the behavior described in UCs into an integrated behavioral model shown as one statechart. Once we decided to use statecharts as the notation in which to unify the UCs, we had to develop a unification method, to apply it in practice, and to evaluate the results. The method, which builds on using statecharts to model UCs and then unifying the UC statecharts into a unified UC statechart [11, 31, 13], is the new OOA method that we called "UCUM". More details on the origin of UCUM are found in the companion paper.

After two full terms in which the students in the course learned and used UCUM to produce DMs for their SRSs, we felt that the quality of the students' DMs had improved. The first author began, for his Ph.D. research, after-the-fact analyses of the DMs and SRSs produced by the students in three consecutive offerings of the CS445 course, the first not using UCUM and the second and third using UCUM, for the purpose of evaluating the quality of UCUM as a method for producing DMs. All the results are reported in the first author's Ph.D. thesis, and many of the qualitative results are reported in the companion paper. As mentioned, this paper focuses on the quantitative evaluation of the repeatability of UCUM for producing DMs. Moreover, it uses the data from the first two of the three consecutive offering of the course. These two offerings were in fact the last offering without UCUM, followed by the first offering with UCUM. Because the only difference, other than

the set of students, between these two offerings was in the OOA method, these two offerings have more similar contexts than do any other pair of non-UCUM-using and UCUM-using offerings.

The repeatability of UCUM and of ordinary OOA for producing DMs were evaluated by after-the-fact analyses of the DMs produced by two consecutive offerings of CS445. Each term's DMs formed one case study:

1. CS O31VS[1], examining 31 large-sized SRSs of a VoIP CBS and its IMS, produced without using UCUM as group term-long projects in one offering of the CS445 class.

2. CS N34VS, examining 34 large-sized SRSs of a VoIP CBS and its IMS, produced using UCUM as group term-long projects in one offering of the CS445 class.

4 Semantic Similarity Analysis

This section describes a general, manually performed procedure to determine in which of two arbitrary sets, A and B, of DMs are the DMs more semantically similar to each other.

The intuition that is simulated by the analysis is best illustrated by showing the two tables of data that told us visually that indeed that the DMs of CS N34VS are slightly more semantically similar to each other than the DMs of the DMs of CS O31VS. Figure 1 shows two tables side by side. The tables have been scaled to make a *visual* impression stand out, the result being that the case study names at the top are barely readable. However, the table on the left side is from CS O31VS and the table on the right side is from CS N34VS. Each row of a table represents one concept, and each column of a table represents one goup's DM. The cell for concept c and DM d is black if and only if c appears in d. The facts that

1. there is slightly more black in the mostly black region at the top of the right table than in the same part of the left table, and

[1]The name of a case study encodes some of the data about the case study; "CS" means "case study", "O" means "Old OOA method with no use case unification", "N" means "New OOA method, i.e., UCUM", a number is the number of SRSs in the case study, and "VS" means "VoIP system".

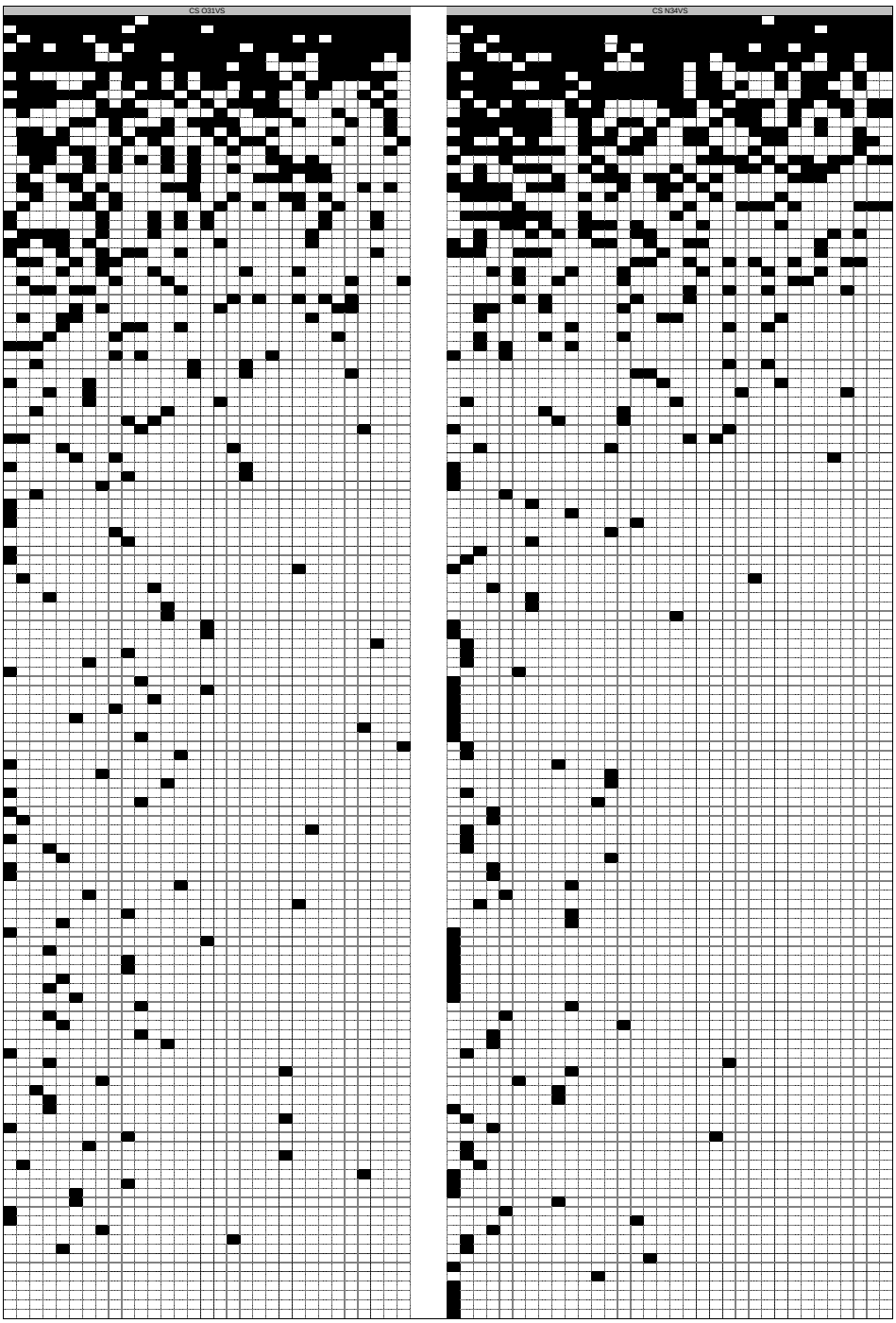

Figure 1. CS O31VS and CS N34VS Data Shown Side-By-Side

2. there is slightly less black in the mostly white region at the bottom of the right table than in the same part of the left table, even when the the black streaks down the first two columns are considered,

say that the DMs of CS N34VS are slightly more semantically similar to each other than the DMs of CS O31VS. Understanding this semantic similarity difference prompted detailed examination of the actual concepts in the DMs of CS O31VS and CS N34VS, which in turn led to the development of the semantic similarity analysis method described in the rest of this section.

Suppose that A and B are sets of DMs that have been constructed to model the same system. The comparison is between between *two* sets of DMs, because each set of DMs has been constructed with a method that is different from that used to construct the other, and the desire is to see if the different construction methods lead to different amounts of semantic similarity in the two sets of DMs. Without loss of generality, the purposes of this procedure are

- to decide whether the DMs contained in A are more semantically similar to each other than the DMs contained in B, and,
- if the models in A are more semantically similar to each other than the models in B, to compute the percentage P by which the models in A are more semantically similar to each other than the models in B.

This procedure has two phases: a data normalization phase, followed by a calculation phase.

The procedure is described as steps to be taken by its performer, who is called upon to make judgements. In fact, for the case studies reported in this paper, the performer was the first author, whose judgement was based on his long experience with the artifacts of the case studies, as described in the beginning of Section 3.

4.1 Phase 1: Data Normalization

DMs that are constructed to model the same problem are likely to have significant semantic overlap, since they are modeling the same conceptual space. At the same time, they are also likely

to exhibit semantic and syntactic variation from each other, since each is created by a different individual or group. The purpose of data normalization is to obtain a normalized view of the concepts in the various DMs of one set of DMs, i.e., to ignore trivial differences in the naming of concepts, in order to be able to show how much the DMs in the set differ, in terms of *semantically unique concepts*.

The name of a concept is taken as its value. Within a single DM, it is assumed that any name refers to a unique concept, e.g., the use of the word "flight" in one travel agency DM is assumed to refer to the same concept throughout that DM. It is assumed also that any one name refers to the same concept in different DMs, e.g., "flight" appearing in two different travel agency DMs is assumed to refer to the same flight concept in both DMs. Finally, it is assumed that all different names from different DMs that refer to what *appear to the analyst* to be the same semantic idea, are nevertheless considered to refer to the same concept, e.g., "flight", "flightInfo", and "flight-Number" are considered to refer to the same concept if the first author believes that what they refer to are the same semantic idea.

To construct a single, representative set of semantically unique concepts in a set D of DMs, first construct the set $RawConcepts(D)$ of all concepts in all of the models in D. Because $RawConcepts(D)$ is a set, *syntactically identical concepts*, i.e., those that have the same name, that appear in more than one DM appear only once in $RawConcepts(D)$.

Next, partition $RawConcepts(D)$ into equivalence classes of elements that refer to the same semantic concept although they may have different names. For example, the three names of the previous example, "flight", "flightInfo", and "flightNumber", would be be in the same equivalence class. For each equivalence class, one member is chosen to be the representative concept name, e.g., "flight" for the equivalence class of the previous example. The determination of equivalences class is performed by comparing the name, attributes, methods, and relationships of each raw concept with those of all other concepts.

Next, for each concept c_{raw} in each DM d in D, replace c_{raw} with the representative concept name c_{rep} for the equivalence class of $RawConcepts(D)$ to which c_{raw} belongs.

Finally, some pruning of the DMs is necessary to reduce the importance of concepts that appear in only one or a few of the DMs. A concept is said to be l-significant if it appears in at least l DMs in D. Then, for some k, remove from all the DMs in D, all of the concepts that are not k-significant. The case studies used 2 as the value of k.

For ease of the discussion, assume, from this point on, that a reference to a concept c belonging to a DM d is a representative concept name rather than the raw concept name that might actually appear in the text of d. Moreover, below, $c(d)$ is defined to be the set of representative concepts appearing in d.

This process might sound very complicated, but it is really just a normalization of different names in different DMs in a set of DMs to a single, unambiguous vocabulary consistent across the set, followed by some simple pruning. The manual process of creating the semantic equivalence classes is labor intensive, slow, and highly subjective. However, it is also likely to be more accurate than any automated approach.

4.2 Phase 2: Determination of Semantic Similarity

The steps for determining, for the sets A and B of DMs, if the DMs of A are more semantically similar to each other than are the DMs of B are:

1. Let D be a set of DMs such that $D = A$ or $D = B$.

   In the following, a, b, and d are DMs such that $a \in A$, $b \in B$, and $d \in D$.

2. For any DM $d \in D$, let $c(d)$ be the set of d's representative concepts.

3. To get the *major concepts* of any DM $d \in D$, for some integer $k \geq 1$, let $mc(d)$ be $c(d)$ restricted to the k-significant concepts.

   By extension, for a set of DMs D, define the *major concepts* of D,

$$mc(D) = \bigcup_{d \in D} mc(d).$$

4. Let

$$cc(A, B) = mc(A) \cap mc(B),$$

i.e., the set of *concepts* that are *common* to A and B, a.k.a. the *common concepts* of A and B.

Note that $cc(A, B)$ is *not* the same as the intersection of the concepts of all DMs in $A \cup B$; that intersection would be a much smaller set of concepts. Instead, $cc(A, B)$ contains all of the concepts that belong to at least k DMs in A and at least k DMs in B.

5. For $d \in A \cup B$, let

$$cc(d) = c(d) \cap cc(A, B),$$

the set of *common concepts* of A and B that are found in DM d.

6. Define

$$cr(d) = \frac{|cc(d)|}{|c(d)|}.$$

This *commonality ratio* measures the number of concepts common to A and B found in a DM d and measures it relative to d's size. This number is close to 1 if d is very similar to the set of common concepts, and is close to 0 if d is very dissimilar to the set of common concepts, i.e., the number is close to 0 if d contains either few common concepts or many uncommon concepts.

7. Define

$$avg_{cc}(D) = \frac{\sum_{d \in D} |cc(d)|}{|D|},$$

the average number of concepts common to A and B found in each d in D.

8. Define

$$avg_{cr}(D) = \frac{\sum_{d \in D} (cr(d))}{|D|},$$

the *average commonality ratio* relative to A and B of each d in D.

9. Now, it is possible to say that the DMs of A are more *semantically similar* to each other than are the DMs of B if and only if

$$avg_{cc}(A) > avg_{cc}(B)$$

and

$$avg_{cr}(A) > avg_{cr}(B)$$

i.e., if and only if, among A and B, the average number of common concepts found in members of A is higher and the *average commonality ratio* of members of A is higher.

10. Finally, if the DMs of A are more semantically similar to each other than are the DMs of B, we define

$$P = avg(avg_{cc}(A)\% - avg_{cc}(B)\%, avg_{cr}(A)\% - avg_{cr}(B)\%),$$

the estimated percentage by which the DMs of A are more semantically similar to each other than are the DMs of B.

For a brief example showing the calculation of all the above defined functions for sets A and B in which each of A and B is a 12-element set of DMs, see Section 5.3 of the first author's Ph.D. thesis [29].

4.3 Additional Optional Phase

In the process of applying the procedure of Section 4.2 to the case studies, we found some additional calculations that can be used to confirm the conclusions of the procedure:

- Let $sdev_{con}(D)$ denote the standard deviation in the number of concepts over a set of DMs.
- Let $sdev_{cc}(D)$ denote the standard deviation in the number of common concepts relative to A and B over a set of DMs.

- Let $iq_{con}(D)$ denote the interquartile range of the number of concepts in a set of DMs.
- Let $iq_{cc}(D)$ denote the interquartile range of the number of common concepts relative to A and B over a set of DMs.

There is additional confirmation that the DMs of A are more semantically similar to each other than are the DMs of B if any of the following is true:

$$sdev_{con}(A) < sdev_{con}(B) \quad iq_{con}(A) < iq_{con}(B)$$
$$sdev_{cc}(A) < sdev_{cc}(B) \quad iq_{cc}(A) < iq_{cc}(B)$$

A more refined estimate of P can be obtained by using any of the following as an additional component of the average defining P:

$$sdev_{con}(B) - sdev_{con}(A) \quad iq_{con}(B) - iq_{con}(A)$$
$$sdev_{cc}(B) - sdev_{cc}(A) \quad iq_{cc}(B) - iq_{cc}(A)$$

5 Analysis of Case Studies

This section discusses the analysis of the two case studies, CS O31VS and CS N34VS. Recall that CS O31VS is about 31 DMs for a VoIP system and its IMS produced *without* the use of UCUM, and CS N34VS is about 34 DMs for the same large-sized CBS produced *with* the use of UCUM.

Table 1 shows the data about the numbers of concepts discovered and captured in all DMs and per DM of CS O31VS and CS N34VS. Notice that there are two columns about CS N34VS, one labeled "CS N34VS" and one labeled "CS N34VS w/DM45". The data for the former column exclude the data for the DM with the most semantically unique concepts, while the data for the latter column include the data for this DM. The excluded DM has 45 concepts and contains a large number of concepts that we were unable to classify and really understand what they represent. This outlier DM is the subject of a later discussion, and it is ignored in the discussion until then, unless it is explicitly mentioned as DM45. As can be seen in Table 1, with this outlier DM45 excluded, the numbers from each DM of CS O31VS and CS N34VS end up being very similar.

Scope	Measure	Case Studies		
		CS O31VS	CS N34VS	CS N34VS w/DM45
In all DMs of the case study	Original, raw concepts	527	622	622
	Syntactically unique concepts	259	312	312
	Semantically unique concepts	134	110	140
In each DM of the case study	Maximum number of semantically unique concepts	31	33	45
	Minimum number of semantically unique concepts	8	10	10
	Average number of semantically unique concepts	17	17	18
	Median number of semantically unique concepts	16	17	17

Table 1. CS O31VS–CS N34VS Statistics

5.1 Detailed Evaluation of Concepts in the DMs of the Case Studies

The main comparison of the semantic similarity of the concept sets of the DMs in CS O31VS and CS N34VS considers the 36 elements of cc((CS O31VS), (CS N34VS w/DM45)), i.e., the concepts common to the DMs of both case studies. Tables 2 and 3 show the distribution of these 36 common concepts in the DMs of CS O31VS CS N34VS, respectively. The 36 common concepts account for to 72% of the major concepts in the DMs in CS O31VS and 77% of the major concepts in the DMs in CS N34VS.

Tables 2 and 3 have the same layout. In each table, each concept c has a row. The row for c is divided into 4 columns, the second of which has subcolumns. The first column contains c's name.[2] The second column is divided into one subcolumn for each DM d. Row c's entry for the subcolumn for d is black if and only if concept c appears in d. The third column contains the number of DMs that have concept c. This number is the number of black subcolumns in the second column of the same row. The fourth column contains the percentage that the number in the third column is of the total number of DMs.

In the third last row of the table, the subcolumn for any d contains the number of common

[2]The concept names in this case study are disguised because at the time of publishing the work, the analyzed project is still being used in the course from which the data come.

Concepts	CS O31VS Common Domain Model Concepts																													# of DMs	% of DMs
	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10	d11	d12	d13	d14	d15	d16	d17	d18	d19	d20	d21	d22	d23	d24	d25	d26	d27	d28	d29	d30	d31		
untuse-33																																30	97
iancal-33																																28	90
onepho-40																																27	87
illbil-41																																26	84
ordcal-34																																25	81
csrcst-42																																23	74
iersys-28																																19	61
gercal-33																																18	58
gerbil-30																																15	48
untcst-34																																11	35
untpho-32																																11	35
gerres-29																																11	35
seruse-41																																11	35
geruse-25																																11	35
germai-26																																10	32
berpho-33																																10	32
olecon-38																																9	29
gerhar-29																																9	29
essip -35																																8	26
roddis-30																																8	26
gerip -35																																7	23
gespri-35																																7	23
gercal-25																																5	16
entpay-38																																5	16
temsys-39																																5	16
logtes-37																																5	16
mertim-40																																5	16
ordern-33																																4	13
gercal-26																																3	10
gercsr-26																																3	10
geraut-23																																2	6
estaut-31																																2	6
rodbil-31																																2	6
gerpri-27																																2	6
ateral-41																																2	6
imetim-41																																2	6
# of Common Concepts	22	19	18	17	17	15	14	14	14	13	13	13	13	12	12	11	11	11	11	11	11	11	10	10	9	9	9	9	8	7	7	Average Commonality Ratio	
# of Concepts	26	24	21	21	20	15	24	22	16	18	15	14	14	18	13	20	19	14	13	12	12	11	17	13	31	16	15	12	20	12	8		
Commonality Ratio	85%	79%	86%	81%	85%	100%	58%	64%	88%	72%	87%	93%	93%	67%	92%	55%	58%	79%	85%	92%	92%	100%	59%	77%	29%	56%	60%	75%	40%	58%	88%		75%

Table 2. CS O31VS Common Concepts

Concepts	CS N34VS Common Domain Model Concepts																																# of DMs	% of DMs
	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10	d11	d12	d13	d14	d15	d16	d17	d18	d19	d20	d21	d22	d23	d24	d25	d26	d27	d28	d29	d30	d31	d32	d33	d34		
lancal-33																																	33	97
untuse-33																																	32	94
lilbil-41																																	31	91
ordcal-34																																	28	82
untpho-32																																	25	74
untcsr-34																																	23	68
genuse-25																																	23	68
gerbil-30																																	22	65
gercal-25																																	22	65
roddis-30																																	18	53
gespri-35																																	18	53
gerhar-29																																	17	50
csrcsr-42																																	16	47
germal-26																																	15	44
essip -35																																	14	41
lensys-28																																	14	41
gercal-33																																	13	38
gerres-29																																	13	38
gercsr-26																																	12	35
entpay-38																																	9	26
gerpri-27																																	9	26
metfrm-40																																	9	26
rodbil-31																																	8	24
gercal-26																																	8	24
onepho-40																																	8	24
logtes-37																																	8	24
imelrn-41																																	7	21
aterat-41																																	5	15
olecon-38																																	4	12
geraut-23																																	2	6
estaut-31																																	2	6
orderr-33																																	2	6
gerip -35																																	2	6
berpho-33																																	2	6
temsys-39																																	2	6
seruse-41																																	2	6
# of Common Concepts	20	20	19	17	17	16	16	16	16	15	15	15	15	14	14	14	14	14	14	13	13	13	13	13	15	14	13	12	11	14	12	11	11	10	Average Commonality Ratio	
# of Concepts	28	22	21	18	18	23	18	17	16	33	19	18	17	45	25	17	17	16	14	18	14	13	13	13	12	14	13	12	12	14	14	11	11	10		
Commonality Ratio	71%	91%	90%	81%	94%	70%	89%	94%	100%	45%	79%	83%	88%	31%	56%	82%	82%	88%	100%	72%	93%	100%	100%	100%	80%	86%	92%	100%	79%	79%	92%	100%	56%	100%		84%

Table 3. CS N34VS Common Concepts

	1	2	3	4	5
1	Metric	CS O31VS Concepts	CS N34VS Concepts	CS O31VS Common Concepts	CS N34VS Common Concepts
2	Average	16.97	17.91	12.29	14.06
3	Standard Deviation	5.07	6.85	3.53	2.60
4	Standard Error	0.91	1.17	0.63	0.45
5	Quartile (.75)	20.00	18.75	14.00	15.75
6	Quartile (.25)	13.00	14.00	10.00	12.00
7	Interquartile Range	7.00	4.75	4.00	3.75

Table 4. CS O31VS–CS N34VS Statistics Summary

concepts in d. In the second last row of the table, the subcolumn for any d contains the number of *all* semantically unique concepts in d. In the last row of the table, the subcolumn for any d contains the commonality ratio for d. Finally, the cell at the intersection of the last row and the last column contains the average commonality ratio for the DMs of the case study, which is the average of all the commonality ratios in the last row.

A visual inspection of the data distribution patterns in Tables 2 and 3 gives an impression similar to that given by the visual inspection of the data distribution patterns in Figure 1. Therefore, additional analysis is needed. Tables 2 and 3 show that

1. on average, 84% of all concepts in the DMs of CS N34VS and

2. on average, 75% of all concepts in the DMs of CS O31VS

are the concepts common to the DMs of both case studies, indicating a higher concentration of common concepts in each DM of CS N34VS than in each DM of CS O31VS.

Table 4 shows that the semantic similarity of common concepts in the DMs of CS N34VS is higher than the semantic similarity of common concepts in the DMs of CS O31VS. Specifically:

1. The average number of concepts per DM is approximately 5.5% higher for the DMs of CS N34VS than for the DMs of CS O31VS, as is shown in the cells in Row 2 and Columns 2 and 3.

2. The average number per DM of common concepts is approximately 14.4% higher for the

DMs of CS N34VS than for DMs of CS O31VS, as is shown in the cells in Row 2 and Columns 4 and 5.

3. For each of the average number of concepts per DM and the average number of common concepts per DM, the standard error of the average for one case study is approximately the same as the standard error of the average for the other case study, due to the approximately equal size of the two sets of data involved.

4. The standard deviation of the average number of concepts per DM is approximately 35.1% higher for the DMs of CS N34VS than for DMs of CS O31VS, as is shown in the cells in Row 3 and Columns 2 and 3.

5. The standard deviation of the average number of common concepts per DM is approximately 35.8% lower for the DMs of CS N34VS than for the DMs of CS O31VS, as is shown in the cells in Row 3 and Columns 4 and 5. The probable reason that the standard deviation of the average number of concepts per DM is higher for the DMs of CS N34VS is the presence in the DMs of CS N34VS of concepts from the outlier DM45. Therefore, we computed also the interquartile range, which can ignore any outlier DM with an extremely large or an extremely small number of concepts.

6. The interquartile range of the number of concepts per DM is approximately 47.4% lower for the DMs of CS N34VS than for the DMs of CS O31VS, as is shown in the cells in Row 7 and Columns 2 and 3.

7. The interquartile range of the number of common concepts per DM is approximately 6.7% lower for the DMs of CS N34VS than for the DMs of CS O31VS, as is shown in the cells in Row 7 and Columns 4 and 5.

That the interquartile range of the number of common concepts per DM is lower in CS N34VS than in CS O31VS for the same set of data leads to the conclusion that the concept concentration was higher in the DMs of CS N34VS than in the DMs of CS O31VS, for both all and the common

concepts. Therefore, the DMs of CS N34VS are more semantically similar to each other than are the DMs of CS O31VS.

Moveover, it is possible to estimate that the semantic similarity of concepts is approximately 10% higher in the DMs of CS N34VS than in the DMs of CS O31VS, based on the previously described quantitative analysis showing that:

- an average increase of 5.5% in the number of concepts per DM of CS N34VS over the number of concepts per DM of CS O31VS
- an average increase of 14.4% in the number of common concepts per DM of CS N34VS over the number of common concepts per DM of CS O31VS
- an increase in the capture of common concepts per DM from 75% in the DMs of CS O31VS to 84% in the DMs of CS N34VS; and
- a narrow data spread for both sets of data.

This 10% improvement in semantic similarity came at a cost. The cost of teaching UCUM and eliciting requirements for the CS N34VS term was at least about 25% higher than the cost of teaching traditional OOA and eliciting requirements for the CS O31VS term. We do not have data on students' hours, but we do have data on hours the TAs interacted with the students. We assume that the students required more time from the TAs in the CS N34VS term than in the CS O31VS term because the students were spending more time on their project at about the same rate. The first author, as the head TA in both terms, was responsible for answering students' questions found that his workload for the CS N34VS term was about 30% higher than for the CS O31VS term. This time is mostly for teaching, since each student was to go to his or her own project TA for elicitation issues. Also, in each term in CS445, we have each TA report his or her actual workload for the course. The average number of elicitation meetings in a term between a group and its TA, as analysts and customer, increased from about 6–8 per previous non-UCUM-using term to about 10 in the UCUM-using term, i.e., from about 66% to about 25% more. That is, using any variant of UCUM required at least about 25% more elicitation effort.

The next step is to perform an in-depth analysis of the results of these two case studies. The focus is on the set of *common* concepts that appeared in the DMs of both case studies. The chart in Figure 2 is built from the data in Tables 2 and 3. Each actual number of groups in CS N34VS was scaled down to maximum of 31, by multiplying it by $31/34$, to match the number of groups in CS O31VS.

For any concept c listed in the x-axis of Figure 2, that c appears less often in the DMs of CS N34VS than in the DMs of CS O31VS suggests either that the use of unified UC statecharts and UCUM prevented c's discovery or that the use of unified UC statecharts and UCUM filtered c out of DMs because c was outside of the domain's boundary. The same number of cs in the DMs of both case studies indicates no change in the effect of the use of unified UC statecharts and UCUM on the discovery of c, and a larger number of cs in the DMs of CS N34VS than in the DMs of CS O31VS indicates that the use of unified UC statecharts and UCUM helped c's discovery.

In the DMs of CS N34VS there were:

- 11 concepts, or 30.6%, that were captured less often than in the DMs of CS O31VS,
- 23 concepts, or 63.9%, that were captured more often than in the DMs of CS O31VS, and
- 2 concepts, or 5.5%, that were captured as often as in the DMs of CS O31VS.

Four concepts of the 11 that appear less often in the DMs of CS O31VS than in the DMs of CS N34VS represent external entities, i.e., Each is an actor from outside the boundary of the VoIP CBS. It was gratifying to see that the DMs of CS N34VS had fewer of the external actors, which should not have appeared, than the DMs of CS O31VS. Nevertheless, it was disappointing that some of external actors still appeared in the DMs of CS N34VS. Thus, building unified UC statecharts for a CBS facilitates but does not guarantee proper boundary definition and proper placement of external actors outside the CBS's boundaries.

The lower numbers of system controller[3] and call manager concepts in the DMs of CS N34VS

[3]The concept names in this discussion are not disguised because the discussion makes no sense without the concept names. We have determined that revealing these particular concept names gives no particular advantage to the students in the course who may have read this paper.

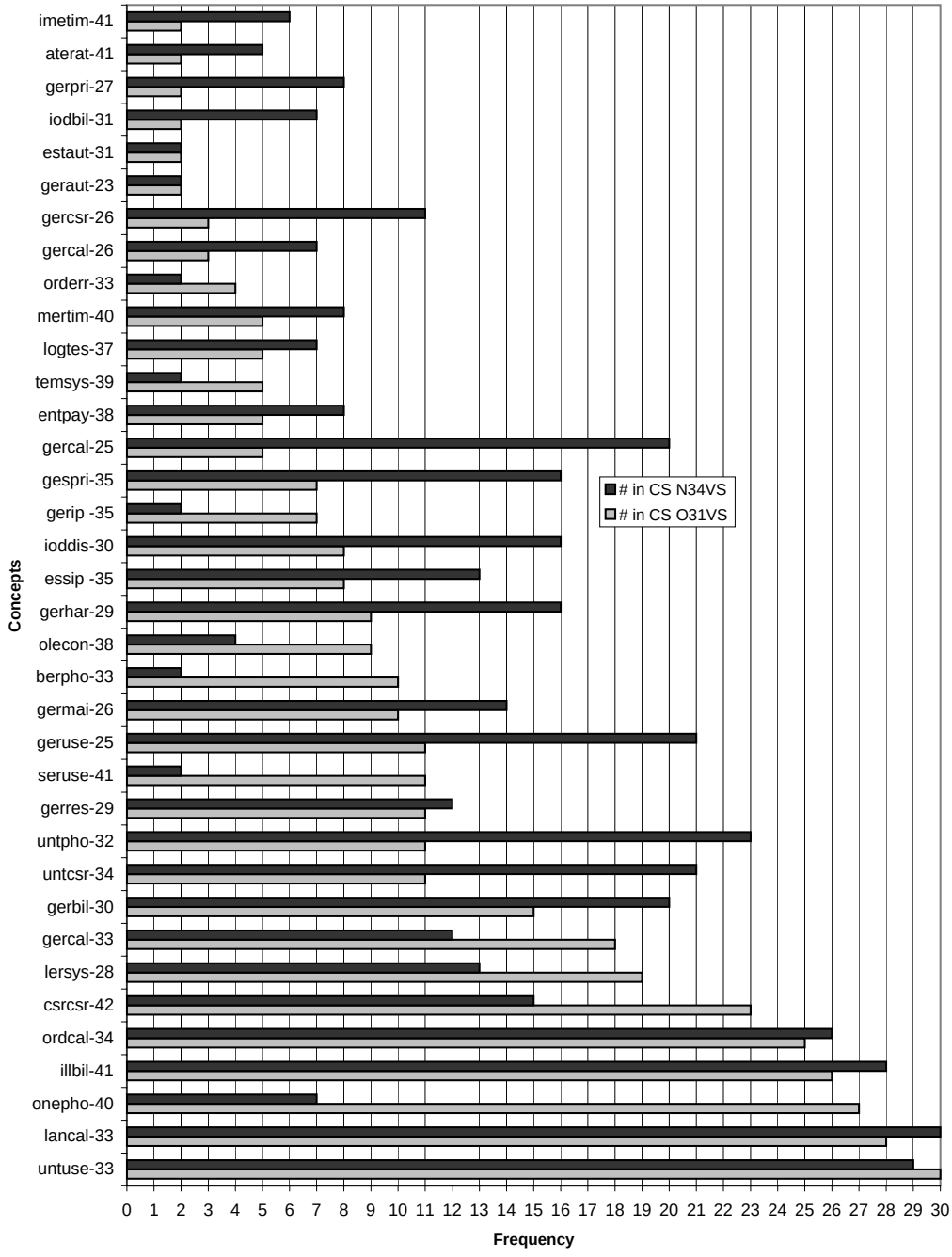

Figure 2. CS O31VS–CS N34VS Common Concepts Chart

than in the DMs of CS O31VS was due to the level of conceptual decomposition and refinement at which DMs in the DMs of CS N34VS were specified; system controller and call manager are the high-level concepts responsible for the overall control of the IMS and of the call processing part of VoIP system, respectively. Because the degree of decomposition in the DMs of CS N34VS is higher than in the DMs of CS O31VS, these concepts tended to be less explicitly indicated in the DMs of CS N34VS than in the DMs of CS O31VS. Instead, they tended to be included indirectly through their components, i.e., the concepts of which they consist.

The system concept represents the CBS itself, and in most cases it was captured through the use of package notation to capture all other concepts within the CBS. This concept is considered redundant because a DM, by definition, captures only concepts *within* the CBS's domain. However, for some reason, some students felt a need to include this concept explicitly as part of the DM.

The chart of Figure 2 shows that the typical concept appears about 10% more often in the DMs of CS N34VS than in the DMs of CS O31VS, in agreement with the estimate that the DMs of CS N34VS are 10% more similar to each other than are the DMs of CS O31VS.

Finally, the outlier DM45, with 30 unclassifiable concepts, taught us about a potentially very negative impact that the use of unified UC statecharts and UCUM can have on DMs, for the specification of any concept that represents only one function of the CBS. Most of the 30 unclassifiable concepts were of this type. For example, a concept such as add admin profile is nothing but one high-level function of CBS captured as a concept. Many a modeling expert considers creating a concept for each function to be incorrect modeling and a negative extreme to which detailed behavioral analysis and modeling can lead. Fortunately, only one out of 34 groups went in this direction. So, the problem was not widespread. The problem could probably have been avoided by the TA's having pointed the group in the right direction.

6 Overall Evaluation and Main Lessons

The analysis of the case studies in Section 5 shows that the semantic similarity of the DMs of CS N34VS was about 10% higher than the semantic similarity of the DMs in CS O31VS. That

is, UCUM is about 10% more repeatable than is ordinary OOA, and the key difference between the methods is that UCUM has an analyst performing a detailed behavioral analysis before doing conceptual analysis. However, the cost of teaching UCUM and eliciting requirements was about 25% higher for the SRSs of CS N34VS than for the SRSs of CS O31VS.

The remainder of this overall evaluation considers observations about UCUM and its products, observations that fall out of the data and our grading evaluation of the artifacts produced by the students.

The two numbers, 10% and 25%, are not comparable and should not be used to make a direct cost–benefit analysis of performing a detailed behavioral analysis before conceptual analysis. For example, one possible benefit of spending the 25% more time eliciting requirements is more detailed requirements and UCs. As described in the companion paper, the TAs and the first author agreed that the requirements and UCs specified in the SRSs of the UCUM-using terms were more detailed and consistent than those of the non-UCUM-using terms. It is necessary also to take the entire lifecycle into account. It is hard, if not impossible, to estimate the total benefits to the downstream development speed and to the reliability, robustness, and other qualities of the CBS being developed caused by the 10% improvement in semantic similarity in the DMs and by the use of UCUM.

The use of UCUM and of extensive functional and behavioral modeling before proceeding with conceptual analysis led to[4]

1. better functional analysis and discovery of more CBS requirements,
2. an increase in semantic similarity among different DMs of the same CBS,
3. several qualitative changes in the DMs:
    - larger number of *functional* concepts, i.e., *processors*,
    - clearer boundary and interface concept definition, and
    - a lack of *inheritance*, and

[4]The first of these results was reported in the companion paper and the rest are reported in Section 5 of this paper.

4. an insufficient number of *data* concepts.

The relationship between the first two points is of particular importance. Consider the detailed behavioral modeling that was done through the use of the unified UC statecharts. This modeling is very similar to, and to some extent, nothing but traditional structured analysis [e.g., 9, 34, 33, 32]. So, rather than contradicting each other, combining structured analysis and OOA helped improve the semantic similarity and the quality of DMs in a way summarized by Richman [23]:

> A weakness of OO is that OO methods only build functional models within the objects. There is no place in the methodology to build a complete functional model. While this is not a problem for some applications (e.g., building a software toolset), for large systems, it can lead to missed requirements. Use cases address this problem, but since all use cases cannot be developed, it is still possible to miss requirements until late in the development cycle.

In addition, it is probably this integration of the detailed behavioral analysis with OOA, in UCUM, that has resulted in DMs that are deeper, more detailed, and closer to the design and architecture of the domain, as observed by John Mylopoulos (in private communication) and some of the graduate students building the DMs of an existing elevator CBS who were familiar with other OOA methods [1, 29]. Mylopoulos observed that the method itself is more systematic than many scenario-based OOA methods. The DMs produced using UCUM seemed to provide a more solid basis for transitioning to design phases than those produced by the OOA methods that had been used previously.

A large number of and well decomposed *functional* concepts, i.e., *processors*, a clear definition of *interface* concepts, and a lack of *inheritance*, are what we consider to be the positive effects of doing a detailed behavioral analysis first on the DMs. In our opinion, such DMs allow easier transition to OOD activities and models.

Each of traditional OOA and UCUM seems unable to expose the *data* concepts in a domain. It seems that no analysis not focusing on data is an adequate substitute for traditional *data* analysis.

Data analysis should be a necessary component of any analysis method and, in our opinion, of a higher priority than conceptual analysis.

In summary, in our opinion, conceptual analysis should not be the main analysis activity. It should be used with behavioral and data analysis. Note that we consider OOD completely distinct from OOA and self-sufficient, and none of the conclusions of this paper apply to OOD.

7 Related Work and Counter Indications

The companion paper describes related work by Glinz; Whittle and Schumann; and Harel, Kugler, and Pnueli [11, 31, 13] whose conclusions generally agree with those of that and this paper.

The present results showing the effectiveness of UCUM as an approach in which an analyst does detailed behavioral analysis *before* doing conceptual analysis contradicts, at least superficially, the conclusions of a case study by Kabeli and Shoval [15]. Their case study shows that doing data modeling before doing functional analysis leads to better OO models, as judged by their criteria [15], than doing functional analysis before doing data modeling. It is not surprising that early case studies produce results that appear to contradict each other. First of all, neither our nor their case study is conclusive, and the two sets of case studies addressed different issues, i.e., neither traditional OOA nor UCUM perform explicit *data* analysis, while the subjects of the Kabeli and Shoval study did explicit *data* analysis. Second, there may be yet other parameters, entirely overlooked in each case study, that consistently account for the contradictory conclusions. Only additional, independent, experimentation in the future can resolve this issue.

8 Conclusion

This paper compares the repeatability of two OOA methods, (1) traditional OOA and (2) UCUM, by comparing the degree of semantic similarity among the sets of DMs for the same problem produced by the two methods. The comparison is based on two after-the-fact case studies:

- Traditional OOA was carried out to produce 31 SRSs of a large VoIP CBS containing an

IMS. Data from these 31 SRSs and their DMs were gathered later in the case study CS O31VS.

- UCUM was carried out to produced 34 SRSs of the same large VoIP CBS containing an IMS. Data from these 34 SRSs and their DMs were gathered later in case study CS N34VS.

The conclusion of the two case studies was that for the CBS, the students, and the SRSs involved in the case studies, the use of UCUM yields DMs that are about 10% more semantically similar to each other than are the DMs yielded by traditional OOA. However, UCUM requires about 25% more time than does traditional OOA.

Other contributions of this work are:

- a discussion of the role of semantic similarity in evaluating a method's repeatability as ar measure of the quality of the method
- a technique to determine in which of two sets of DMs are the DMs more semantically similar to each other, and
- an in-depth analysis of the quantitative results to obtain ideas for further improvements of OOA methods.

Acknowledgments

We thank all students and TAs who were in the classes in which UCUM was developed and applied; we thank in particular those who provided feedback. We thank also Ric Holt, John Mylopoulos, and Ladan Tahvildari, the independent reviewers of this research work for their comments and suggestions.

Davor Svetinovic's work was supported in parts by Canadian NSERC Postgraduate Scholarship PGS B-255929-2002 and a Canadian FCAR Doctoral Scholarship. Daniel Berry's, Michael Godfrey's, and Nancy Day's work was supported in part by Canadian NSERC Grant Numbers NSERC-RGPIN227055-00, NSERC-RGPIN217236-99, and NSERC-RGPIN240537-01, respectively.

References

[1] CS846 course project. http://se.uwaterloo.ca/~dberry/ATRE/ElevatorSRSs/; accessed January 30, 2006.

[2] Semantic similarity definition. http://en.wikipedia.org/wiki/Semantic_similarity; accessed September 15, 2006.

[3] Turnstile system. http://swag.uwaterloo.ca/~dsvetinovic/turnstilesystem.pdf; accessed January 30, 2006.

[4] Grady Booch. Object-oriented design. *Ada Lett.*, I(3):64–76, 1982. ISSN 1094-3641.

[5] Grady Booch. *Object-Oriented Analysis and Design with Applications*. Benjamin/Cummings, Redwood City, CA, USA, second edition, 1994.

[6] Rolv Bræk and Øystein Haugen. *Engineering real time systems: an object-oriented methodology using SDL*. Prentice Hall International, 1993. ISBN 0-13-034448-6.

[7] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. Addison-Wesley, Boston, Massachusetts, first edition, 1995.

[8] Ole-Johan Dahl, Bjorn Myhrhaug, and Kristen Nygaard. *SIMULA 67 Common Base Language*. Norwegian Computing Centre, Oslo, Norway, 1968.

[9] Tom DeMarco. *Structured Analysis and System Specification*. Yourdon Press, New York, 1978.

[10] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. John Wiley & Son Ltd, Hoboken, N.J., first edition, 1996.

[11] Martin Glinz. An integrated formal model of scenarios based on statecharts. In *Proceedings of the 5th European Software Engineering Conference*, pages 254–271, London, UK, 1995. Springer-Verlag. ISBN 3-540-60406-5.

[12] David Harel. Statecharts: A visual formalism for complex systems. *Sci. Comput. Program.*, 8(3):231–274, 1987. ISSN 0167-6423.

[13] David Harel, Hillel Kugler, and Amir Pnueli. Synthesis revisited: Generating statechart models from scenario-based requirements. In *Lecture Notes in Computer Science*, volume 3393 of *LCNS*, pages 309–324. Springer-Verlag, January 2005.

[14] Les Hatton. Does OO really match the way we think? *IEEE Software*, 15(3):46–54, 1998.

[15] Judith Kabeli and Peretz Shoval. Data modeling or functional analysis: What comes next? an experimental comparison using FOOM methodology. In *Procedings of the Eighth CAISE–IFIP WG 8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design (EMMSAD'03)*, pages 48–57, 2003.

[16] Hermann Kaindl. Is object-oriented requirements engineering of interest? *Requirements Engineering*, 10(1):81–84, 2005.

[17] Jeff Kramer. Abstraction: The key to software engineering? *Keynote: JSSST Japan Society for Software Science and Technology Conference*, 2004.

[18] John Krogstie, Odd Ivar Lindland, and Guttorm Sindre. Towards a deeper understanding of quality in requirements engineering. In JuhanI Iivari, Kalle Lyytinen, and Matti Rossi, editors, *Proceedings of Seventh International Conference on Advanced Information Systems Engineering (CAiSE'95)*, pages 82–95. Springer, 14–16 June 1995.

[19] John Krogstie, Guttorm Sindre, and Håvard Jørgensen. Process models representing knowledge for action: A revised quality framework. *European Journal of Information Systems*, 15 (1):91–102, February 2006.

[20] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process*. Prentice Hall, Englewood Cliffs, NJ, second edition, 2001.

[21] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process*. Prentice Hall, Englewood Cliffs, NJ, second edition, 2001.

[22] Odd Ivar Lindland, Guttorm Sindre, and Arne Sølvberg. Understanding quality in conceptual modeling. *IEEE Software*, 11(2):42–49, 1994.

[23] Dale M. Rickman. A process for combining object oriented and structured analysis and design. In *Proceedings of NDIA 3rd Annual Systems Engineering & Supportability Conference*, October 2000.

[24] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, Reading, MA, second edition, 2004.

[25] Isabel Santos and Jo ao Álavaro Carvalho. An assessment of the applicability of object-oriented analysis to the development of information systems. In *Proceedings of the IEEE International Conference on Requirements Engineering (ICRE)*, pages 238–244, April 1996.

[26] SE463. SE463/CS445 course project. `http://www.student.cs.uwaterloo.ca/~cs445/`; accessed January 30, 2006.

[27] Sally Shlaer and Stephen J. Mellor. *Object-oriented systems analysis: modeling the world in data*. Yourdon Press, 1988. ISBN 0-13-629023-X.

[28] Guttorm Sindre. Private communication, 2006.

[29] Davor Svetinovic. *Increasing the Semantic Similarity of Object-Oriented Domain Models by Performing Behavioral Analysis First*. PhD thesis, University of Waterloo, 2006.

[30] Davor Svetinovic, Daniel M. Berry, Nancy A. Day, and Michael W. Godfrey. Unified use case statecharts: Case studies. *Requirements Engineering Journal*, to appear, 2007.

[31] Jon Whittle and Johann Schumann. Generating statechart designs from scenarios. In *ICSE '00: Proceedings of the 22nd International Conference on Software Engineering*, pages 314–323, New York, NY, USA, 2000. ACM Press. ISBN 1-58113-206-9.

[32] Roel Wieringa. A survey of structured and object-oriented software specification methods and techniques. *ACM Comput. Surv.*, 30(4):459–527, 1998. ISSN 0360-0300.

[33] Edward Yourdon. *Modern Structured Analysis*. Prentice Hall, Englewood Cliffs, NJ, 1988.

[34] Edward Yourdon and Larry Constantine. *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*. Prentice Hall, Englewood Cliffs, NJ, 1979.